INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(3)954.90.20

Rapports Techniques

N° 40

NEW USERS' INTRODUCTION TO QNAP2

Dominique POTIER

Octobre 1984

NEW USERS'

INTRODUCTION TO QNAP2

Dominique Potier

(version: 08/20/84)

Abstract

This document is an introduction to the modelling package QNAP2 based upon simple modelling examples of computer system performance. The examples are presented in progressive manner, in order to illustrate the main features of QNAP2 and its application to a large range of modelling problems.

Résumé

Ce rapport constitue une introduction au logiciel de modélisation QNAP2 construite à partir d'exemples simples de modélisation des performances de systèmes informatiques. Les exemples sont présentés de manière progressive, afin d'illustrer les aspects les plus importants de QNAP2, et son application à une large classe de problèmes de modélisation.

CONTENTS

12 A communication network model

Preface

This document is an introduction to QNAP2 based upon simple modelling examples of computer systems. The models have been chosen in order to illustrate the main features of QNAP2 rather than the art of computer system modelling. Thus, throughout the document it is assumed that the reader is familiar with the fundamentals of the usage of queueing network models for the performance analysis of computer systems. The various modelling assumptions involved in the models presented here are only briefly summarized but are not discussed.

This document shows how a model can be built and analysed using the facilities provided by QNAP2. For a detailed and complete description of these facilities the reader should refer to the QNAP2 Reference Manual.

Introduction

2.1/ QNAP2 objectives

Modelling and evaluation of computer systems performance have benefited by an important research effort in the recent years. This effort has contributed to major theoretical advances in methods and tools, and to the elaboration of a general methodology for the performance modelling of computer architectures and installations. This methodology is based on the representation of a computer system as a network of queues and servers. In this framework, the servers of the network represent the physical and logical processors of the system, and the entities cycling in the network stand for the programs or the processes being executed and competing for these resources. The value of this representation has been validated in many instances, and this methodology is now regarded as operationnal.

Research efforts in the field of modelling and evaluation have also resulted in a wide spectrum of efficient resolution techniques for queueing network models having complementary ranges of utilization. These methods differ by their cost of operation and by the limitations they impose on the generality of the model being studied. Each method realizes a certain compromise between cost, accuracy and applicability. It is thus important to be able to have access to these different techniques in order to select the one which represent the best compromise for the problem to be analysed.

However, these techniques are based on different theoretical bases, and, practically, the application of several resolution methods on a given model is a long and cumbersome task. In the absence of specific aids, the non-specialist will have to restrict his attention to one method (the one he knows and masters) and to discard the other approaches.

It is thus essential to provide computer systems designers and users with a tool which facilitates the practical use of these various methods. This is the main objective of the package QNAP2 (Queueing Network Analysis Package 2) which is presented here.

The main features of QNAP2 are covered in the next section. Various examples illustrate these features in all the following sections.

2.2/ QNAP2 main features

QNAP2 (Queueing Network Analysis Package 2) is a software tool for describing and solving queueing network models. QNAP2 consists of :

- a collection of resolution algorithms, including discrete event simulation , exact and approximate mathematical methods,

- a common user interface for model description, analysis control, and result presentation.

A modelling study may involve various heterogeneous methods (for example, in the analysis of a large complex system, system modules requiring a detailed representation are analysed by simulation, and partial results obtained may be afterwards combined into a global mathematical model of the whole system). QNAP2 enables the analyst to use the same interface language for all the modelling process. One thus disposes of a tool for the quantitative analysis of computer systems which not only allows system designers and users to have access to sophisticated analysis techniques (the more recent research results are included in QNAP2), but also enables the concurrent and combined application of these techniques to a given problem.

The QNAP2 package provides the analyst with a framework for modelling computer systems and with the techniques to solve the models. However, the analysis of a real system in order to reach a "satisfactory" model and the validation of this model (in order to check its predicting quality) remain under the sole control and responsability of the analyst. As indicated below, the aid provided by QNAP2 in a modelling process resides mainly in the stage MODEL -> SOLUTION. The stages SYSTEM -> MODEL and SOLUTION -> SYSTEM have to be carried out by the analyst (see Figure 1).

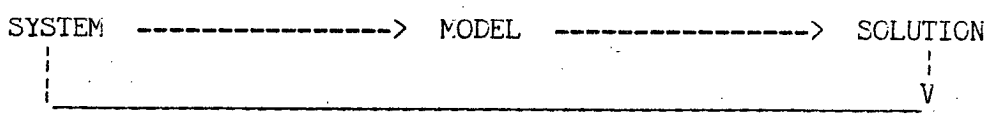

Figure 1: modelling process

/END/

STOP

3.1/ Presentation

In this section we will consider the detailed description and analysis of a simple example. In doing so, we will illustrate some of the basic features of QNAP2 introduced in section 2 and a step by step approach to QNAP2.

The model considered here is a simple queueing network model of a transaction processing system as represented in Figure 2.

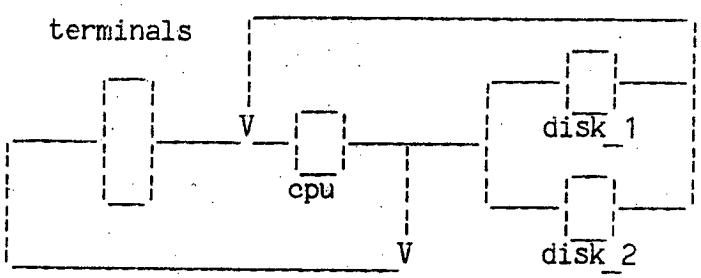

Figure 2: transaction processing system

The basic components of a queueing network model are stations and customers. There are four stations in the example of Figure 2:

- the set of terminals (station terminal),
- the central processing unit (station cpu),
- the two disk-units (stations disk_1 and disk_2).

In the example customers represent transactions. The path followed by a transaction through the stations of the network during its life-time is represented by routing rules: where a transaction can go when it leaves a station and how it decides where to go. The routing rules used in the model are probabilistic rules. This means that any feasible transition of a transaction from one station of the model to an other is characterized by a transition probability.

We assume that a transaction performs on the average 3 disk_1 accesses and 6 disk_2 accesses before terminating. This behavior is represented in the model by assigning a transition probability of 0.3 to the transition cpu ——> disk_1, a transition probability of 0.6 to the transition cpu ——> disk_2, and a transition probability of 0.1 to the transition cpu ——> terminal.

A transaction's service time represents the uninterrupted cpu time allocated to the transaction (and possibly associated processes) during each visit to the cpu (i.e. the virtual cpu time between two consecutive physical disk accesses). Actually a transaction's service time will be the sum of several times including overheads, actual processing time of the transaction and others. We assume that a transaction's service time at the cpu has an hyper-exponential distribution independent of the current state of the cpu with squared coefficient of variation equal to 5.

In the same fashion, the various times involved in a disk service time are aggregated into one service time having an exponential distribution.

The queueing discipline at the cpu is processor-sharing (PS) in order to ensure a fair treatment to all the transactions. The queueing discipline at the disk_1 and disk_2 is first-in first-out (FIFO).

The station terminal represent the set of terminals connected to the system. We assume that there are as many terminals as users, so that there is no waiting of transactions at a terminal. The behavior of the users at their terminal is characterized by their thinking time, i.e. the time from receiving the output of the previous transaction to submitting a new transaction.

3.2/ Description of the model

3.2.1/ The user interface

The QNAP2 user interface is a specific language comprised of control statements or commands, and algorithmic statements. A QNAP2 user may use this language in two ways: he or she may prepare a QNAP2 program, i.e. an ordered sequence of commands, using his or her standard text editor and have this program processed by QNAP2; he or she may interact directly with QNAP2 and have commands and statements translated and processed in an interactive mode. The most usual and practical approach is to use a combination of those two modes: a QNAP2 program describing the model under study is built off-line; the analysis and modifications of the initial model are performed on-line. From now on we will follow this approach and assume that the QNAP2 user is logged on a time-sharing system and has access to a text editor.

3.2.2/ Declaration of objects and variables

Building a queueing network model consists in assembling complex items such as queues, servers, resources and customers so as to represent the organisation and behavior of the physical system studied. In QNAP2 most of these items will be defined as objects. An object is a structured variable made up of scalar elements (INTEGER, REAL, BOOLEAN or STRING), references to other objects, arrays or other objects. These elements are call the attributes of the object. The set of attributes of an object and the operators which may be applied to it define the type of the object.

There are four predefined object types in QNAP2: QUEUE, CUSTOMER, CLASS and FLAG. Within this example we shall restrict our attention to the object type QUEUE. We will show how the object type QUEUE is defined and the way to specify the service station associated with an object of type QUEUE.

The first step in building a QNAP2 program is to declare the objects and variables which will be used. The declaration statements must follow a /DECLARE/ command (note that all QNAP2 commands are introduced by a keyword enclosed within two slash characters and that the first / of a command must be the first non-blank character of a line). For the example of Figure 2, we write the following declaration statement:

```
/DECLARE/ QUEUE terminal, cpu, disk_1, disk_2;
   INTEGER nb_term;
   REAL t_cpu, t_term, t_disk_1, t_disk_2;
```

In QNAP2 a station is composed of a unique queue: consequently a station will be referenced by the name of its queue and before describing the stations of the model one must declare the queues associated to these stations (cpu, terminal, disk_1, disk_2) as objects of type QUEUE. QUEUE is a predefined object type so that all the objects declared with the type QUEUE will possess the attributes and the properties of this type of objects.

The two other declaration statements declare an INTEGER scalar variable nb_term representing the number of terminals and a set of REAL scalar variables representing the mean service time of the transaction in the stations.

Note that all the keywords of QNAP2 use uppercase letters. For sake of clarity all the user defined identifiers of the examples will be writen in lower case letters (the underscore character is part of the character set of the language).

Also, the user may want to comment his program. This is advisable, especially if the model is complex and is to be used intensively, and because only the first eight characters of a QNAP2 identifier are significant. A comment starts with the character & and includes all the following characters up to the end of the physical source line. The previous declaration statements will look like this after comments have been added:

```
/DECLARE/                  & declaration statement
&
QUEUE
        cpu,               & central processing unit
        terminal,          & set of terminals
        disk_1,            & disk unit number 1
        disk_2;            & disk unit number 2
&
INTEGER
        nb_term;           & number of terminals
&
REAL
        t_cpu,             & mean service time at cpu
        t_disk_1,          & mean service time at disk_1
        t_disk_2,          & mean service time at disk_2
        t_term;            & mean think time at terminals
```

3.2.3/ Declaration of new QUEUE attributes

From this straightforward description we may note the two following observations. First a real variable is associated with each declared queue: hence this real variable can as well be considered as a new attribute of the object type QUEUE; then we declared two queues disk_1 and disk_2 to represent the queues associated with the two disk stations: a more general and powerfull description can be performed by considering an array of queues whose dimension will be specified according to the model considered.

We thus build a new description of the model taking into account these two observations. First we cancel the previous one using the command /RESTART/ which clears the data and workspace associated with the current description.

```
/RESTART/
```

Then we define a new attribute t for the object type QUEUE with the following declaration statement:

```
/DECLARE/ QUEUE REAL t;
```

and declare the queues of the model as before. Because of the previous attribute declaration statement each declared queue will now have a new real scalar attribute t.

```
/DECLARE/                  & declaration statement
&
INTEGER n_disk = 2;        & number of disk units
&
REAL    trans(n_disk);     & visit ratios to disk units
&
QUEUE
        cpu,               & central processing unit
        terminal,          & set of terminals
        disk (n_disk);     & disk units
&
INTEGER
        nb_term;           & number of terminals
```

As noted before, an array of queues, disk (n_disk), has been declared with n_disk = 2 (the dimension of the array must be known at declaration time because the elements of the array are then physically created). However, as we will see, the rest of the description is fully parametrized by the parameter n_disk. The integer array, trans, contains the visit ratios of the transactions to the disk units.

3.2.4/ Definition of the stations of the model

The next important command is the command /STATION/. The function of this command is to describe one or several stations of the model. /STATION/ is a parametrized command, the parameters being used to specify the properties and characteristics of the stations (name, type, scheduling, service...). The following is a possible description of the cpu station:

```
/STATION/ NAME = cpu;
          SCHED = PS;
          SERVICE = HEXP(t, 5.);
          TRANSIT = disk(1 STEP 1 UNTIL n_disk), trans(1 STEP 1 UNTIL n_disk),
                    terminal, 1;
```

The value of the first parameter NAME is the identifier of the queue associated with the station being described. As noted above, this identifier must have been declared with the type QUEUE. NAME is the one mandatory parameter of the command /STATION/ and must appear in the first place. The order of the other parameters is not significant.

The next parameter appearing in the description of the cpu station is the SCHED parameter. The value of this parameter is a keyword (FIFO, LIFO, PS, PRIOR...) indicating the scheduling algorithm (or queueing discipline) for the station.

The third parameter in the description of the cpu station is the SERVICE parameter. The value specified must be a statement of the algorithmic language: either a simple statement (for instance a work demand procedure such as EXP( ) or CST( ) or a synchronization procedure such as P( ) or WAIT( )), either a compound statement (BEGIN...END block).

The service time at the cpu is represented in QNAP2 by the work demand procedure HEXP(t ,5.). Work demand procedures are special procedures which can be used only in a SERVICE parameter and which define the probability distribution of the amount of work required by a customer during one visit to the station. The work demands are expressed in appropriate work units, e.g. micro-sec., seconds, instructions, bytes...In the example SERVICE = HEXP(t_cpu, 5.) means that the service times at the cpu are independent identically distibuted hyperexponential random variables with mean t_cpu and squared coefficient of variation 5.

Also it should be noted that in the parameter list of the procedure call HEXP(t ,5), t represents the attribute of the queue associated with the station being defined , i.e. cpu.

The fourth parameter appearing in the description of the station cpu is the TRANSIT parameter. The TRANSIT parameter describes the routing rules of transactions at service completion time. The transition probabilities may as well be specified as relative frequencies or visit ratios. The expression of the parameter TRANSIT in the example makes use of the list facilities of QNAP2: it is comprized of two couples of sublists, each couple containing a queue sublist and a real sublist. The meaning of this expression is that the visit ratio to disk(i) is trans(i), for i=1,...n_disk, and that the visit ratio to the station terminal is 1.

The disk stations have similar structures: they will be defined witin the same /STATION/ command as follows:

```
/STATION/ NAME = disk( 1 STEP 1 UNTIL n_disk);
          SERVICE = EXP(t);
          TRANSIT = cpu;
```

Note that the queueing discipline has not been explicitly specified: in order to simplify the description of models, QNAP2 uses default options whenever possible. The default value of SCHED is FIFO.

This example illustrates one the facilities provided by QNAP2 for describing a set of similar stations. Two different machanisms are combined here: the attribute mechanism through the attribute t and the list mechanism within the parameter NAME. Hence the disk stations specified have simular structure (number of servers, scheduling,...), but different parameters (mean service time).

The terminal station definition is similar to the other definitions, except that we introduce two new parameters: TYPE and INIT.

```
/STATION/ NAME = terminal;
          TYPE = INFINITE;
          SERVICE = EXP(t);
          INIT = nb_term;
          TRANSIT = cpu;
```

TYPE is used to specify the type of station being considered: single server station, multiple server station, semaphore or resource station. TYPE has the default value SINGLE, SERVER which means that if the parameter TYPE has not been specified the station will be considered as a single server station (thus, cpu, disk_1 and disk_2 are single server stations).

The terminal station is represented in QNAP2 by the type INFINITE, which indicates that the station has always a server (a terminal) for each transaction in the station.

The INIT parameter indicates the initial number of transactions in the station terminal. In other words, since there are as many terminals as transactions, the value of INIT represents the number of active terminals connected to the system.

3.3/ Analysis of the model

3.3.1/ Definition of specific analysis options

Having shown the description of the model, we now show how the model can be analysed and solved. First of all, we may define an entry block and an

exit block. These blocks contain statements which will be executed respectively just before and immediatly after the analysis is performed. In our example we will define an entry block in order to edit the parameters of the model.

An entry block (or an exit block) is defined within the command /CONTROL/ with the parameter ENTRY (or EXIT) (the general function of the command /CONTROL/ is to control the way the analysis and solution of a model is performed). The following shows a possible entry block:

```
/DECLARE/ REF QUEUE r_q;
&
/CONTROL/ ENTRY = BEGIN
                  PRINT(" ");
                  PRINT("input parameters");
                  PRINT(" ");
                  PRINT("number of terminals        : ",nb_term);
                  FOR r_q:= ALL QUEUE DO
                  PRINT("mean service time at ", r_q, " : ",r_q.t );
                  END;
```

The queue reference variable r_q is defined in order to manipulate the objects of type QUEUE previously defined. The expression ALL QUEUE generates the list of all the created queues and for all the elements of this list the name of the queue and its associated mean service time will be printed.

Note that the function of an entry block could be as well implemented by a user defined procedure containing the same statements and invoked just before a solver is requested.

3.3.2/ Definition of the analyses

The model initialization and execution is invoked by the command /EXEC/. The purpose of the /EXEC/ command is to introduce and execute a statement (simple statement or compound statement) of the algorithmic language. Following is an /EXEC/ command for the model. The first statements give values to the numerical parameters of the model (the values given for times are in units of seconds). SOLVE is a procedure which calls for the set of the analytical solvers of QNAP2 and solves the model selecting the most appropriate solver.

```
/EXEC/ BEGIN
       nb_term:= 5;
       cpu.t:= 0.2;
       disk(1).t:= 0.08; disk(2).t:= 0.1;
       trans:= 3, 6;
       terminal.t:= 30.;
       SOLVE;
       END;
```

The execution of the SOLVE procedure will result in the execution of the ENTRY block and then in the analysis of the model described with the most appropriate solver. Because no specific output option is specified the results of the analysis will be edited in a standard report containing the basic performance criteria for each station of the model.

input parameters

```
number of terminals          :          5.
mean service time at cpu     :     .2000
mean service time at terminal :    30.00
mean service time at disk   1 :   0.8000E-01
mean service time at disk   2 :     .1000
- MEAN VALUE ANALYSIS ("MVA") -
*************************************************************************
*  NAME     *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*           *          *          *          *          *          *
* cpu       * .2000    * .2988    * .3852    * .2579    * 1.494    *
*           *          *          *          *          *          *
* terminal  * 30.00    *0.0000E+00* 4.481    * 30.00    * .1494    *
*           *          *          *          *          *          *
* disk   1  *0.8000E-01*0.3585E-01*0.3691E-01*0.8236E-01* .4481    *
*           *          *          *          *          *          *
* disk   2  * .1000    *0.8963E-01*0.9645E-01* .1076    * .8963    *
*           *          *          *          *          *          *
*************************************************************************
      MEMORY USED:       3062 WORDS OF 4 BYTES
       (  1.17  % OF TOTAL MEMORY)
```

As indicated in the standard report output after the analysis has been performed, QNAP2 has selected the Mean Value Analysis method to solve the model studied. Indeed, for this model mean value analysis is the most efficient approach in term of computational complexity, memory requirements and accuracy.

3.3.3/ Contents of the standard report

The default option is to have the standard report printed when the analysis is completed. The standard report gives, for each station, the following standard performance criteria:

utilization factor (BUSYPCT):

the utilization factor of a station is the percentage of busy servers in the station. It is equal to zero in the case of an infinite station (since the number of servers is infinite, the utilization of each server is zero).

mean service time (SERVICE):

this criteria is the mean service time as seen by the customers.

mean response time (RESPONSE):

the response time is defined as the sum of the waiting time and service time.

mean number of customer (CUST NB):

the mean number of customers in the station includes customers being served or waiting.

mean throughput (THRUPUT):

the mean throuput of a station is defined as the mean number of customer services completed per time unit.

3.3.4/ Definition of specific outputs

From the standard report, the performance of the system can be analysed. The main observation is that the load of the system is rather low with the present configuration: cpu utilization is 0.2988, disk 1 and disk 2 utilizations are under 0.1. Equilibrium conditions imply that the throughput of the system is equal to the troughput of the station terminal: 0.1494 transactions processed per second.

These results concern performance criteria defined with respect to each station. Obviously, other criteria may be meaningfull as, for example in this model, the mean response time of transactions, i.e. the mean time from leaving the terminals to returning to the terminals. A standard way to compute this criteria is to apply the Interactive Response Time Formula of Operational Analysis:

mean response time = number of transactions / throughput - mean thinking time

Using this rule, we can compute directly the mean response time of transactions from the figures of the standard report. We get:

mean response time = 5. / 0.1494 - 30. = 3.472

Also, we may program this rule and output the result in an exit block. In this way we will get the mean response time of transactions every time an analysis of the model is performed.

In order to program the Interactive Response Time Formula we use the QNAP2 result access functions. These functions return the standard performance criteria computed as the result of an analysis. For instance, the mean throughput of the station terminal will be returned by MTHRUPUT (terminal).

Working in the interactive mode of QNAP2, we input the following command:

```
/CONTROL/ EXIT = BEGIN
                 PRINT(" ");
                 PRINT("results");
                 PRINT(" ");
                 PRINT("mean response time of trans. : ",
                 nb_term/MTHRUPUT(terminal)-terminal.t);
                 END;
```

Now, we need no more the standard report. This may be specified by the OPTION parameter of the /CONTROL/ command. Since we are still within the /CONTROL/ command, we type:

```
OPTION = NRESULT;
```

(although the output of the standard report has been cancelled, this report could still be printed on demand using the QNAP2 procedure OUTPUT as follows: /EXEC/ OUTPUT;)

Finally, we can request a new analysis. Note that we do not have to initialize the numerical parameters of the model: they have been initialized by the execution of the previous /EXEC/ command in the QNAP2 program, and those initializations are still active.

```
 /EXEC/ SOLVE;

input parameters

number of terminals        :        5.
mean service time at cpu       :    .2000
mean service time at terminal  :   30.00
mean service time at disk  1   :  0.8000E-01
mean service time at disk  2   :    .1000

results

mean response time of trans. :    3.472
```

3.3.5/ Definition of a macro-statement

Suppose we wish now to evaluate the performance of the transaction system for a variety of numbers of terminals and mean cpu service times. We can do this by typing an /EXEC/ command, setting the numerical parameters and calling the SOLVE procedure. However, this may be somewhat tedious if we have to repeat it many times. It is more efficient to define a QNAP2 macro-instruction as follows:

```
$MACRO eval(arg1,arg2)
/EXEC/ BEGIN
       nb_term:= arg1;
       cpu.t:= arg2;
       SOLVE;
       END;
$END
```

3.3.6/ Evaluation of different modelling assumptions

A call to the macro_instruction $eval(x1, x2) will be equivalent to the input of the whole macro definition body, in which the dummy arguments arg1 and arg2 are replaced by the actual parameters x1 and x2. For example:

```
$eval(5,0.25)

input parameters

number of terminals        :        5.
mean service time at cpu       :    .2500
mean service time at terminal  :    30.00
mean service time at disk   1  :   0.8000E-01
mean service time at disk   2  :    .1000

results

mean response time of trans. :    4.316


$eval(10,0.25)

input parameters

number of terminals        :       10.
mean service time at cpu       :    .2500
mean service time at terminal  :    30.00
mean service time at disk   1  :   0.8000E-01
mean service time at disk   2  :    .1000

results

mean response time of trans. :    6.626


$eval(15,0.25)

input parameters

number of terminals        :       15.
mean service time at cpu       :    .2500
mean service time at terminal  :    30.00
mean service time at disk   1  :   0.8000E-01
mean service time at disk   2  :    .1000

results

mean response time of trans. :    11.57
```

These evaluations show the influence of the number of active terminals on the mean response time of the system. Also, we can investigate the effect of a variety of design assumptions. For instance, the impact of a FIFO scheduling policy at cpu is obtained with the following statements (note that any parameter of a station may be modified in the same way).

```
/STATION/ NAME = cpu;
          SCHED = FIFO;
```

```
$eval(5,0.25)

input parameters

number of terminals          :        5.
mean service time at cpu     :     .2500
mean service time at terminal :    30.00
mean service time at disk  1 :  0.8000E-01
mean service time at disk  2 :     .1000

results

mean response time of trans. :    5.400
```

```
$eval(10,0.25)

input parameters

number of terminals          :       10.
mean service time at cpu     :     .2500
mean service time at terminal :    30.00
mean service time at disk  1 :  0.8000E-01
mean service time at disk  2 :     .1000

results

mean response time of trans. :    9.113
```

As it is well known from queueing theory, FIFO scheduling tends to degrade the performance if service time distributions with large coefficients of variation are involved.

It is interesting to recall that in this example the scheduling at cpu is FIFO and the service time distribution is not exponential. Solution methods based on the product form theorems are no longer applicable. In order to solve this model QNAP2 selected an approximate solver, called ITERATIVE.

The impact of a bi-processor cpu is specified by:

```
 /STATION/ NAME = cpu;
           TYPE = MUTIPLE(2);
                  !
==>ERROR (STATION) : UNKNOWN PARAMETER OR OPTION ... MUTIPLE
```

The keyword MULTIPLE has been incorrectly typed. QNAP2 gives an error message indicating that an unknown parameter has been entered. When such an error occurs, QNAP2 proceeds with the next command of the input text. Thus, in the interactive mode, QNAP2 expects a new command:

```
 /STATION/ NAME = cpu;
           TYPE = MULTIPLE(2);


 $eval(5,0.25)

input parameters

number of terminals        :         5.
mean service time at cpu         :    .2500
mean service time at terminal    :    30.00
mean service time at disk   1    :   0.8000E-01
mean service time at disk   2    :    .1000
==>ERROR (SOLVE) : NO ANALYTICAL METHOD CAN BE APPLIED
            MODIFY THE MODEL OR TRY ANOTHER METHOD
```

No appropriate analytical method was found and the request resulted in an error message. The reason is that in the present version of QNAP2 there is no analytical solver applicable to queueing network models involving a multi-server station with FIFO scheduling and non-exponential service times.

Then, we can either invoke an other solver (for example SIMUL or MARKOV), either modify the model assuming again that the scheduling at cpu is processor-sharing (PS). First, we choose to invoke the Markovian solver of QNAP2. This solver will analyze the model description, generate the states of the model and its matrix of transition probabilities, and solve the matrix in order to obtain the steady-state solution. The option NRESULT is cancelled in order to get the standard report.

```
/CONTROL/ OPTION = RESULT;
&
/EXEC/ MARKOV;
```

```
input parameters

number of terminals        :         5.
mean service time at cpu       :    .2500
mean service time at terminal  :    30.00
mean service time at disk   1  :   0.8000E-01
mean service time at disk   2  :    .1000

***  MARKOVIAN ANALYSIS ***


TOTAL NUMBER OF STATES =       131
NUMBER OF NON-ZERO ELEMENTS IN MATRIX =      1081

    13 ITERATIONS
*************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*          *          *          *          *          *          *
* cpu      * .2500    * .1869    * .3814    * .2551    * 1.495    *
*          *          *          *          *          *          *
* terminal * 30.00    *0.0000E+00* 4.485    * 30.00    * .1495    *
*          *          *          *          *          *          *
* disk   1 *0.8000E-01*0.3588E-01*0.3698E-01*0.8245E-01* .4485    *
*          *          *          *          *          *          *
* disk   2 * .1000    *0.8969E-01*0.9682E-01* .1079    * .8969    *
*          *          *          *          *          *          *
*************************************************************************
      MEMORY USED:     123922 WORDS OF 4 BYTES
       ( 47.50  % OF TOTAL MEMORY)

results

mean response time of trans. :    3.446
```

The report indicates the number of states of the model (note that non-exponential distributions are treated as Coxian distributions and that an additional state represent the current step in the service) and the number of non-zero elements in the matrix of transition probabilities.

Then we modify the scheduling at the cpu station and assume again that a processor-sharing policy is used. A sequence of three analyses using the FOR statement is requested:

```
/CONTROL/ OPTION = NRESULT;
&
/STATION/ NAME = cpu;
          SCHED = PS;
&
/EXEC/ FOR nb_term := 5, 10, 15 DO SOLVE;
```

```
input parameters

number of terminals        :        5.
mean service time at cpu       :    .2500
mean service time at terminal  :   30.00
mean service time at disk   1  :  0.8000E-01
mean service time at disk   2  :    .1000

results

mean response time of trans. :   3.435

input parameters

number of terminals        :       10.
mean service time at cpu       :    .2500
mean service time at terminal  :   30.00
mean service time at disk   1  :  0.8000E-01
mean service time at disk   2  :    .1000

results

mean response time of trans. :   3.732

input parameters

number of terminals        :       15.
mean service time at cpu       :    .2500
mean service time at terminal  :   30.00
mean service time at disk   1  :  0.8000E-01
mean service time at disk   2  :    .1000

results

mean response time of trans. :   4.293
```

With a bi-processor cpu, no degredation of the response time is observed when the number of terminals connected to the system is increased from 5 to 15.

3.4/ Termination of a session

A QNAP2 session is terminated by the command /END/ which causes a STOP message:

```
/END/
STOP
```

4.1/ Presentation

It was implicitly assumed in the last section that all the transactions processed by the system had similar behaviours. This is obviously a simplistic view of the workload of the system, and a more detailed characterization is necessary in most situations. Thus, we now assume that three classes of transactions have been identified: the two first classes correspond to foreground transactions issued by interactive users accessing two different data-base stored respectively on disk_1 and disk_2; the third class consists of background transactions performing administration and statistical work on the two data-bases. Those background transactions are issued from a specific terminal.

The behaviour and routing rules of the three classes of transactions are represented in the Figures 3, 4 and 5.

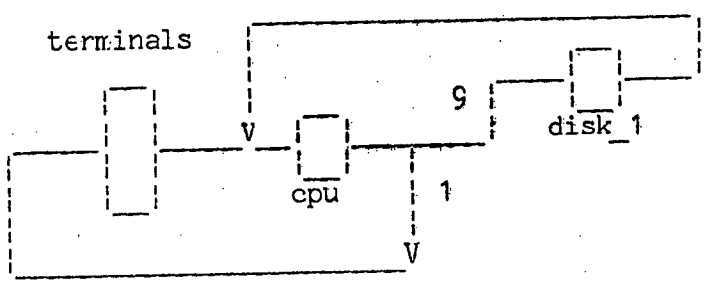

Figure 3: database 1 transactions

Transactions accessing database 1 perform on the average 9 accesses to disk_1 before terminating. They have exponential service times at the cpu.

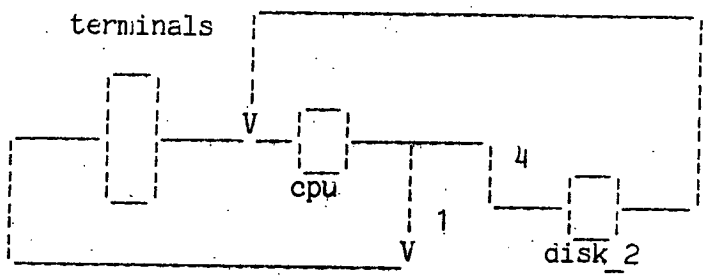

Figure 4: database 2 transactions

Transactions accessing database 2 perform on the average 4 accesses to disk_2 before terminating. They have exponential service times at the cpu.

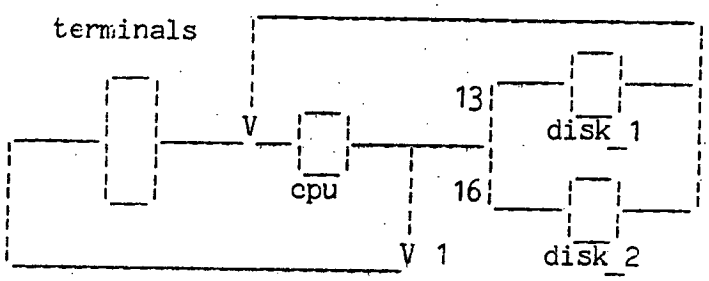

Figure 5: background transactions

Background transactions perform on the average 13 accesses to disk_1 and 16 accesses to disk_2 before terminating. They have exponential services times at the cpu.

The service times of transactions at the disk_1 and disk_2 is assumed independent of the type of transaction performing the i/o.

4.2/ Description of the model

4.2.1/ The object type CLASS

In order to represent the concept of classes of transactions we introduce the predefined object type CLASS. The object type CLASS is used to handle populations of customers having the same behavior in terms of service time distributions, priorities, routing rules,...As the other object types of QNAP2 the object type CLASS can be enhanced with new attributes.

Several modifications of the first model have to be made in order to take into account these different classes of transactions. First, we define new attributes to the object type CLASS:

```
/DECLARE/
&
INTEGER n_disk = 2;          & number of disk units
&
CLASS INTEGER nb_term;       & number of terminals for a class
CLASS REAL t_cpu,            & mean service time at cpu for a class
          t_term;            & mean thinking time for a class
CLASS STRING cl_name;        & class name
CLASS REAL trans(n_disk);    & visit ratios to disk units
```

The three classes of transactions are then declared:

```
CLASS
        dbase_1,        & class of foreground transactions
                        & accessing data base 1
        dbase_2,        & class of foreground transactions
                        & accessing data base 2
        admin;          & class of background transactions
```

and the queues associated with the stations of the model:

```
QUEUE REAL t;
&
QUEUE
        cpu,            & central processing unit
        terminal,       & set of terminals
        disk(n_disk);   & disk units
```

Note that in the exemple the real queue attribute t will only be used to characterize the mean service times of transactions at the disk units. For the other stations the mean service times is contained in the attributes of the object CLASS. This point is illustrated by the definition of the stations.

4.2.2/ Definition of the stations

```
/STATION/ NAME = cpu;
          SCHED = PS;
          SERVICE (ALL CLASS) = EXP(t_cpu);
          TRANSIT (ALL CLASS) =
                 disk(1 STEP 1 UNTIL n_disk), trans(1 STEP 1 UNTIL n_disk),
                 terminal, 1;
```

As shown in this description, the service times and routing rules are defined for each class of transaction. The expression ALL CLASS is equivalent to the list (d_base_1, dbase_2, admin). The service description means that the service times of transactions of class dbase_1, dbase_2 and admin have exponential random variables with respective means dbase_1.t_cpu, dbase_2.t_cpu and admin.t_cpu (an object attribute is referenced by prefixing t identifier by the object identifier, or by a reference to it).

Note that if a parameter of the /STATION/ command is not indexed by a class ide the value of the parameter is assigned to all the currently declared classes. As a consequence the service description of the station cpu could have been defined as:

```
/STATION/ NAME = cpu;
          SERVICE = EXP(t_cpu);
```

An evaluation of the service expression would then have been performed for all the declared classes.

This is also illustrated by the description of the disk stations.

```
/STATION/ NAME = disk(1 STEP 1 UNTIL n_disk);
          SERVICE  = EXP(t);
          TRANSIT = cpu;
```

The parameters of the station terminal remain unchanged. However, the description is again generic since t_term and nb_term are class attributes:

```
/STATION/ NAME = terminal;
          TYPE = INFINITE;
          SERVICE = EXP(t_term);
          INIT = nb_term;
          TRANSIT = cpu;
```

4.3/ Analysis of the model

4.3.1/ Definition of analysis options

In order to write the entry block we need a variable referencing a class. Variables referencing objects are called references and are declared by associating the keyword REF to the object type identifier of the objects being referenced:

```
/DECLARE/ REF CLASS r_class; INTEGER i;
&
/CONTROL/ ENTRY = BEGIN
                  PRINT(" ");
                  PRINT("input parameters");
                  PRINT(" ");
                  FOR i:= 1 STEP 1 UNTIL n_disk DO
                  PRINT("mean service time at ",disk(i),": ",disk(i).t);
                  FOR r_class:= ALL CLASS DO
                  BEGIN
                  PRINT(" ");
                  PRINT(r_class.cl_name);
                  PRINT("number of terminals        : ",r_class.nb_term);
                  PRINT("mean service time at cpu   : ",r_class.t_cpu);
                  PRINT("mean thinking time         : ",r_class.t_term);
                  END;
                  END;
&
/CONTROL/  EXIT = BEGIN
                  PRINT(" ");
                  PRINT("results");
                  PRINT(" ");
                  PRINT("cpu utilization          : ",MBUSYPCT(cpu));
                  FOR i:= 1 STEP 1 UNTIL n_disk DO
                  PRINT("utilization of ",disk(i),": ",MBUSYPCT(disk(i)));
                  FOR r_class:= ALL CLASS DO
                  BEGIN
                  PRINT(" ");
                  PRINT(r_class.cl_name);
                  PRINT("throughput                 : ",
                  MTHRUPUT(terminal,r_class));
                  IF MTHRUPUT(terminal,r_class) > 0. THEN
                  PRINT("mean response time         : ",
                  r_class.nb_term/MTHRUPUT(terminal,r_class)-r_class.t_term);
                  END;
                  END;
&
/CONTROL/ OPTION = NRESULT; CLASS = ALL QUEUE;
```

The control parameter CLASS = ALL QUEUE; requests that the performance criteria are computed per class for all the declared stations. This control is mandatory in our example because the throughput of each class of transaction is needed to compute their respective response time.

4.3.2/ Analysis of different modelling assumptions

```
/EXEC/ BEGIN
       &
       dbase_1.cl_name:= "data-base 1";
       dbase_2.cl_name:= "data-base 2";
       admin.cl_name:= "administrator";
       &
       dbase_1.nb_term:= 5;
       dbase_2.nb_term:= 10;
       admin.nb_term:= 1;
       dbase_1.trans:=9,0;
       &
       dbase_1.t_term:= 20.;
       dbase_2.t_term:= 30.;
       admin.t_term:= 120.;
       dbase_2.trans:= 0,4;
       &
       dbase_1.t_cpu:= 0.2;
       dbase_2.t_cpu:= 0.1;
       admin.t_cpu:= 1.5;
       admin.trans:= 13,16;
       &
       disk(1).t:= 0.12; disk(2).t:= 0.18;
       &
       END;
&
```

We are now ready to interact with the model. An interesting point to evaluate is the impact of the background transactions on the response time of foreground transactions. This can be evaluated by analyzing the performance of the system with and without active background transaction:

```
/EXEC/ BEGIN
       admin.nb_term:= 0;
       SOLVE;
       END;

 input parameters

 mean service time at disk   1 :    .1200
 mean service time at disk   2 :    .1800

 data-base 1
 number of terminals         :        5.
 mean service time at cpu    :    .2000
 mean thinking time          :   20.00

 data-base 2
 number of terminals         :       10.
 mean service time at cpu    :    .1000
 mean thinking time          :   30.00

 administrator
 number of terminals         :        0.
 mean service time at cpu    :   1.500
 mean thinking time          :   120.0

 results

 cpu utilization           :    .5583
 utilization of disk   1   :    .2168
 utilization of disk   2   :    .2258

 data-base 1
 throughput                :    .2008
 mean response time        :   4.904

 data-base 2
 throughput                :    .3136
 mean response time        :   1.889

 administrator
 throughput                :  0.0000E+00
```

```
/EXEC/ BEGIN
       admin.nb_term:= 1;
       SOLVE;
       END;
```

```
input parameters

mean service time at disk   1  :    .1200
mean service time at disk   2  :    .1800

data-base 1
number of terminals          :        5.
mean service time at cpu     :    .2000
mean thinking time           :   20.00

data-base 2
number of terminals          :       10.
mean service time at cpu     :    .1000
mean thinking time           :   30.00

administrator
number of terminals          :        1.
mean service time at cpu     :   1.500
mean thinking time           :   120.0

results

cpu utilization              :    .7443
utilization of disk   1      :    .2135
utilization of disk   2      :    .2365

data-base 1
throughput                   :    .1910
mean response time           :   6.172

data-base 2
throughput                   :    .3100
mean response time           :   2.256

administrator
throughput                   :  0.4604E-02
mean response time           :   97.22
```

The results show that the background transactions have a detrimental effect on the response time of foreground transactions. This was expected because of the additional load due to the background transactions, and also because of the processor-sharing (PS) scheduling policy at the cpu which results in equal allocation of the cpu resource to all the classes of transactions.

One way to improve the response time of foreground transactions is to assign priority levels to each class of transaction and a priority queueing discipline to the cpu. This is done as follows:

```
/STATION/ NAME = cpu;
          SCHED = PRIOR, PREEMPT;
          PRIOR(dbase_1) = 3;
          PRIOR(dbase_2) = 2;
          PRIOR(admin) = 1;
```

Increasing levels of priorities have been assigned to the classes of transactions. The class admin has the lowest level of priority; the class dbase_1 the highest. The analysis is now requested in the usual way:

```
/EXEC/ SOLVE;
```

```
input parameters

mean service time at disk   1  :    .1200
mean service time at disk   2  :    .1800

data-base 1
number of terminals            :        5.
mean service time at cpu       :    .2000
mean thinking time             :   20.00

data-base 2
number of terminals            :       10.
mean service time at cpu       :    .1000
mean thinking time             :   30.00

administrator
number of terminals            :        1.
mean service time at cpu       :   1.500
mean thinking time             :   120.0
```

results

```
cpu utilization          :    .7535
utilization of disk  1   :    .2298
utilization of disk  2   :    .2302

data-base 1
throughput               :    .2068
mean response time       :   4.181

data-base 2
throughput               :    .3029
mean response time       :   3.014

administrator
throughput               :  0.4190E-02
mean response time       :   118.7
```

The results show the decrease of the response times of the foreground transactions due to the priority scheduling. They are computed by a specific solver of QNAP2 called PRIORPR. This solver is selected when the model studied contains one station with priority scheduling.

The standard report gives a detailed account of the performance of the system:

```
 /EXEC/ OUTPUT;
*************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*          *          *          *          *          *          *
* cpu      * .2032    * .7535    * 1.706    * .4602    * 3.708    *
*(dbase_1 )* .2000    * .4135    * .5933    * .2869    * 2.068    *
*(dbase_2 )* .1000    * .1514    * .6393    * .4221    * 1.514    *
*(admin   )* 1.500    * .1885    * .4736    * 3.768    * .1257    *
*          *          *          *          *          *          *
* terminal * 26.71    *0.0000E+00* 13.73    * 26.71    * .5139    *
*(dbase_1 )* 20.00    *0.0000E+00* 4.135    * 20.00    * .2068    *
*(dbase_2 )* 30.00    *0.0000E+00* 9.087    * 30.00    * .3029    *
*(admin   )* 120.0    *0.0000E+00* .5028    * 120.0    *0.4190E-02*
*          *          *          *          *          *          *
* disk   1 * .1200    * .2298    * .2796    * .1460    * 1.915    *
*(dbase_1 )* .1200    * .2233    * .2713    * .1458    * 1.861    *
*(admin   )* .1200    *0.6536E-02*0.8296E-02* .1523    *0.5446E-01*
*          *          *          *          *          *          *
* disk   2 * .1800    * .2302    * .2890    * .2261    * 1.279    *
*(dbase_2 )* .1800    * .2181    * .2737    * .2259    * 1.212    *
*(admin   )* .1800    *0.1207E-01*0.1532E-01* .2286    *0.6703E-01*
*          *          *          *          *          *          *
*************************************************************************
```

An other way to represent the flow of background transactions is to consider that these transactions are issued from an infinite source (as opposed to the finite source considered in the previous description). In QNAP2, a source is represented by a station having the type SOURCE (TYPE = SOURCE;). The value of the SERVICE parameter defines the intervals of times between two consecutive generation of customers. The TRANSIT parameter specifies the destination station(s) and class(es) of the generated transaction.

The previous model is completed by declaring a queue source, defining the corresponding station (the attribute source.t represents the mean time between two consecutive generations of transaction), and by redefining the TRANSIT parameter of the station cpu for the class admin:

```
/EXEC/ admin.nb_term:= 0;
&
/DECLARE/ QUEUE source;
&
/STATION/ NAME = source;
          TYPE = SOURCE;
          SERVICE = EXP(t);
          TRANSIT = cpu, admin;
&
/STATION/ NAME = cpu;
          TRANSIT(admin) = disk(1 STEP 1 UNTIL n_disk),
                           trans(1 STEP 1 UNTIL n_disk),
                           OUT, 1;
```

The last destination station specified in the parameter TRANSIT for transactions of class admin is OUT: OUT is a predefined queue (i.e. OUT has the object type QUEUE) which acts as a sink. Transactions sent to the OUT queue are destroyed.

Also, we have to redefine the ENTRY and EXIT blocks of the program because the structure of the source station for admin transactions has changed and the Interactive Response Time Formula is not applicable to transactions of class admin:

```
/DECLARE/ REAL x; REF QUEUE r_q;
&
/CONTROL/ ENTRY = BEGIN
                  PRINT(" ");
                  PRINT("input parameters");
                  PRINT(" ");
                  FOR i:= 1 STEP 1 UNTIL n_disk DO
                  PRINT("mean service time at ",disk(i),": ",disk(i).t);
                  FOR r_class:= dbase_1, dbase_2 DO
                  BEGIN
                  PRINT(" ");
                  PRINT(r_class.cl_name);
                  PRINT("number of terminals        : ",r_class.nb_term);
                  PRINT("mean service time at cpu   : ",r_class.t_cpu);
                  PRINT("mean thinking time         : ",r_class.t_term);
                  END;
                  PRINT(" ");
                  PRINT(admin.cl_name);
                  PRINT("mean service time at cpu   : ",admin.t_cpu);
                  PRINT("mean inter-arrival time    : ",source.t);
                  END;
```

```
EXIT = BEGIN
       PRINT(" ");
       PRINT("results");
       PRINT(" ");
       PRINT("cpu utilization          : ",MBUSYPCT(cpu));
       FOR i:= 1 STEP 1 UNTIL n_disk DO
       PRINT("utilization of ",disk(i),": ",MBUSYPCT(disk(i)));
       FOR r_class:= dbase_1, dbase_2 DO
       BEGIN
       PRINT(" ");
       PRINT(r_class.cl_name);
       PRINT("throughput               : ",
       MTHRUPUT(terminal,r_class));
       IF MTHRUPUT(terminal,r_class) > 0. THEN
       PRINT("mean response time       : ",
       r_class.nb_term/MTHRUPUT(terminal,r_class)-r_class.t_term);
       END;
       PRINT(" ");
       PRINT(admin.cl_name);
       PRINT("throughput               : ",
       MTHRUPUT(source));
       IF MTHRUPUT(source) > 0. THEN
       BEGIN
       x:= 0;
       FOR r_q:= disk, cpu DO x:= x+ MCUSTNB(r_q, admin);
       PRINT("mean response time       : ", x/MTHRUPUT(source));
       END;
       END;
```

(the response time of background transactions is computed using Little Rule)

We can now request the analysis of the model after the mean inter-arrival time of the admin transactions has been specified:

```
/EXEC/ BEGIN
       source.t:= 250.;
       SOLVE;
       END;
```

```
input parameters

mean service time at disk   1  :    .1200
mean service time at disk   2  :    .1800

data-base 1
number of terminals          :       5.
mean service time at cpu     :    .2000
mean thinking time           :   20.00

data-base 2
number of terminals          :      10.
mean service time at cpu     :    .1000
mean thinking time           :   30.00

administrator
mean service time at cpu     :   1.500
mean inter-arrival time      :   250.0
==>ERROR (SOLVE) :  NO ANALYTICAL METHOD CAN BE APPLIED
            MODIFY THE MODEL OR TRY ANOTHER METHOD
```

The request resulted in an error message indicating that none of the analytical solvers available through SOLVE can solve the model. The reason is that the field of application of the analytical solver PRIORPR specialized in the analysis of priority queues is restricted to closed models.

In order to solve the model with an analytical solver, some assumption has to be modified, for instance the scheduling at the cpu station:

```
/STATION/ NAME = cpu;
          SCHED = PS;
```

```
/EXEC/ SOLVE;

input parameters

mean service time at disk  1 :    .1200
mean service time at disk  2 :    .1800

data-base 1
number of terminals          :       5.
mean service time at cpu     :    .2000
mean thinking time           :   20.00

data-base 2
number of terminals          :      10.
mean service time at cpu     :    .1000
mean thinking time           :   30.00

administrator
mean service time at cpu     :   1.500
mean inter-arrival time      :   250.0

results

cpu utilization          :    .7146
utilization of disk  1   :    .2114
utilization of disk  2   :    .2343

data-base 1
throughput               :    .1899
mean response time       :   6.325

data-base 2
throughput               :    .3094
mean response time       :   2.317

administrator
throughput               : 0.4000E-02
mean response time       :   136.9
```

4.3.3/ Analysis of the model by simulation

An other way to solve the open model without having to modify the scheduling at the cpu is to use simulation rather than analytical techniques. Thus, we may come back to the former specification of the station cpu:

```
/STATION/ NAME = cpu;
          SCHED = PRIOR, PREEMPT;
          PRIOR(dbase_1) = 3;
          PRIOR(dbase_2) = 2;
          PRIOR(admin) = 1;
```

The analysis of a model by simulation is invoked by the procedure SIMUL, in the same fashion as the analytical solvers are invoked by the procedure SOLVE. However, we must provide some additional information to define the simulation run. This information concerns:

- the simulation length,
- the confidence interval estimation,
- the tracing options.

The definition of the simulation length is mandatory. The other definitions are optional and will be discussed in the next sections. All these definitions are made within the /CONTROL/ command through specific parameters. The parameter used to define the simulation length is TMAX: the value of TMAX may be a real constant or expression. The parameter used to request the estimations of confidence intervals on the standard performance criteria is ACCURACY: the value of ACCURACY is the list of queues for which confidence intervals are requested.

Three different methods are implemented in QNAP2 for the derivation of confidence intervals: the replication method, the regeneration method and the spectral method. These methods are introduced in section 5. The default method used to estimate the confidence intervals is the spectral method.

```
/DECLARE/ REAL max_time;
&
/CONTROL/  TMAX = max_time; ACCURACY = cpu, disk;
```

Also, the EXIT blpck is updated in order to edit the confidence intervals. Specific result access functions are available to retrieve the estimated confidence intervals.

```
EXIT = BEGIN
       PRINT(" ");
       PRINT("results");
       PRINT(" ");
       PRINT("cpu utilization          : ",MBUSYPCT(cpu),
        " +/- ",CBUSYPCT(cpu));
       FOR i:= 1 STEP 1 UNTIL n_disk DO
       PRINT("utilization of ",disk(i),": ",MBUSYPCT(disk(i)),
        " +/- ",CBUSYPCT(disk(i)));
       FOR r_class:= dbase_1, dbase_2 DO
       BEGIN
       PRINT(" ");
       PRINT(r_class.cl_name);
       PRINT("throughput                : ",
       MTHRUPUT(terminal,r_class));
       IF MTHRUPUT(terminal,r_class) > 0. THEN
       PRINT("mean response time        : ",
       r_class.nb_term/MTHRUPUT(terminal,r_class)-r_class.t_term);
       END;
       PRINT(" ");
       PRINT(admin.cl_name);
       PRINT("throughput                : ",
       MTHRUPUT(source));
       x:= 0;
       IF MTHRUPUT(source) > 0. THEN
       FOR r_q:= disk, cpu DO x:= x+ MCUSTNB(r_q, admin);
       PRINT("mean response time        : ", x/MTHRUPUT(source));
       END;
```

We will simulate the system during 12500 seconds so that 50 background transactions or so are processed:

```
/EXEC/ BEGIN
       max_time:= 12500.;
       SIMUL;
       END;
```

input parameters

```
mean service time at disk   1  :     .1200
mean service time at disk   2  :     .1800

data-base 1
number of terminals          :         5.
mean service time at cpu     :     .2000
mean thinking time           :    20.00

data-base 2
number of terminals          :        10.
mean service time at cpu     :     .1000
mean thinking time           :    30.00

administrator
mean service time at cpu     :    1.500
mean inter-arrival time      :    250.0
```

results

```
cpu utilization              :     .7168     +/-   0.5510E-01
utilization of disk   1      :     .2254     +/-   0.1003E-01
utilization of disk   2      :     .2302     +/-   0.1029E-01

data-base 1
throughput                   :     .2084
mean response time           :    3.992

data-base 2
throughput                   :     .3054
mean response time           :    2.748

administrator
throughput                   :  0.3520E-02
mean response time           :    136.3
```

5.1/ Presentation

We implicitly assumed in the models presented in the last two sections that transactions need no other resource than the processor and the disk units to be processed. This is obviously a simplification since passive resources such as main memory and channels are also involved in the operation of the system and may have a severe impact on the global performance if contentions occur on these resources. In this section we show how passive resources can be dealt with QNAP2.

Thus we assume that each transaction operates in a memory partition (or buffer), and that there are a limited number of partitions available. A transaction entering the system first requests a memory partition: if a partition is available, the transaction occupies this partition and then proceeds as described in the previous sections; if no memory partition is available the transaction is blocked and waits until a partition is released by an other transaction. Once a transaction has obtained a partition and completed its processing, it releases the partition it occupied and quits the system.

This behaviour is described in Figure 6: in_q and out_q are two passive stations where transactions request and release their memory partitions. The memory partitions and the queue of blocked transactions are represented by a passive station with as many passive servers as memory partitions.

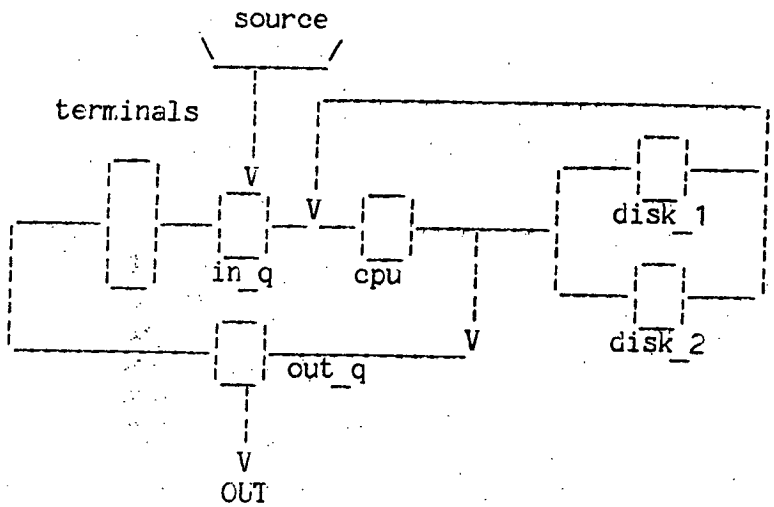

Figure 6: model with memory contention

5.2/ Description of the model

5.2.1/ Definition of a resource station

We start with the model defined in the last section and complete the description by the three new stations memory, in_q and out_q:

```
/DECLARE/ QUEUE
&
          memory,       & set of memory partitions
          in_q,         & entry queue
          out_q;        & exit queue
&
          INTEGER
&
          buffer;       & number of memory buffers
&
/STATION/ NAME = memory;
          TYPE = RESOURCE, MULTIPLE (buffer);
&
/STATION/ NAME = in_q;
          TYPE = INFINITE;
          SERVICE = P(memory);
          TRANSIT = cpu;
&
/STATION/ NAME = out_q;
          TYPE = INFINITE;
          SERVICE = V(memory);
          TRANSIT(dbase_1, dbase_2) = terminal;
          TRANSIT(admin) = OUT;
```

The first /STATION/ command defines the station memory as a resource station with a number of resource units equal to buffer. In stations in_q and out_q transactions request and release a memory partition by executing the procedures P and V on the resource memory; once these operations have been completed transactions proceed to the next station according to the possible transitions specified in the TRANSIT parameter.

5.2.2/ Modification of the previous definitions

The routing rules at stations cpu, terminal and source must be modified due to the introduction of the stations in_q and out_q in the queueing network:

```
/STATION/ NAME = cpu;
          TRANSIT(dbase_1, dbase_2) = disk(1 STEP 1 UNTIL n_disk),
                                      trans(1 STEP 1 UNTIL n_disk), out_q, 1;
          TRANSIT(admin) =  disk(1 STEP 1 UNTIL n_disk),
                            trans(1 STEP 1 UNTIL n_disk), out_q, 1;
/STATION/ NAME = terminal;
          TRANSIT(dbase_1, dbase_2) = in_q;
&
/STATION/ NAME = source;
          TRANSIT = in_q, admin;
```

We now update the entry and exit blocks.

```
/CONTROL/ ENTRY = BEGIN
                  PRINT(" ");
                  PRINT("input parameters");
                  PRINT(" ");
                  FOR i:= 1 STEP 1 UNTIL n_disk DO
                  PRINT("mean service time at ",disk(i),": ",disk(i).t);
                  PRINT("number of memory partitions: ", buffer);
                  FOR r_class:= dbase_1, dbase_2 DO
                  BEGIN
                  PRINT(" ");
                  PRINT(r_class.cl_name);
                  PRINT("number of terminals        : ",r_class.nb_term);
                  PRINT("mean service time at cpu   : ",r_class.t_cpu);
                  PRINT("mean thinking time         : ",r_class.t_term);
                  END;
                  PRINT(" ");
                  PRINT(admin.cl_name);
                  PRINT("mean service time at cpu   : ",admin.t_cpu);
                  PRINT("mean inter-arrival time    : ",source.t);
                  END;
    EXIT = BEGIN
           PRINT(" ");
           PRINT("results");
           PRINT(" ");
           PRINT("cpu utilization :  ",MBUSYPCT(cpu),
            " +/- ",CBUSYPCT(cpu));
           FOR i:= 1 STEP 1 UNTIL n_disk DO
           PRINT("utilization of ",disk(i),":  ",MBUSYPCT(disk(i)),
            " +/- ",CBUSYPCT(disk(i)));
           FOR r_class:= dbase_1, dbase_2 DO
           BEGIN
           PRINT(" ");
           PRINT(r_class.cl_name);
           PRINT("mean blocked time :  ",
           MBLOCKED(in_q, r_class));
           PRINT("throughput :  ",
           MTHRUPUT(terminal,r_class));
           IF MTHRUPUT(terminal,r_class) > 0.  THEN
           PRINT("mean response time :  ",
           r_class.nb_term/MTHRUPUT(terminal,r_class)-r_class.t_term);
           END;
           PRINT(" ");
           PRINT(admin.cl_name);
           PRINT("mean blocked time :  ",
           MBLOCKED(in_q,admin));
           PRINT("throughput :  ",
           MTHRUPUT(source));
```

```
x:= 0;
IF MTHRUPUT(source) > 0.  THEN
FOR r_q:= in_q, memory DO x:= x+ MCUSTNB(r_q, admin);
PRINT("mean response time :  ", x/MTHRUPUT(source));
END; &.qne
```

The exit block has been completed in order to print the mean time spent by blocked transactions in station in_q. This criteria is retrieved by the result access function MBLOCKED. Note also that in the computation of the mean response time of admin transactions using Little Rule, we use the fact that the number of transactions in the system is, by definition of station memory, always equal to the number of active transactions in the resource station memory. Indeed, resource stations provide a simple mean to obtain the standard performance criteria related to a sub-network of the network studied. In the example the performance criteria computed for memory give valuable information on the behavior of the system comprised of the stations cpu, disk_1 and disk_2.

5.3/ Analysis of the model

The model is analysed with the assumption that the maximum multiprogramming level is 5.

```
/EXEC/ BEGIN
       buffer:= 5;
       SIMUL;
       END;
```

```
input parameters

mean service time at disk   1  :    .1200
mean service time at disk   2  :    .1800
number of memory partitions:         5.

data-base 1
number of terminals          :         5.
mean service time at cpu     :    .2000
mean thinking time           :   20.00

data-base 2
number of terminals          :        10.
mean service time at cpu     :    .1000
mean thinking time           :   30.00

administrator
mean service time at cpu     :   1.500
mean inter-arrival time      :   250.0
```

Resources

results

cpu utilization		:	.6888	+/- 0.6434E-01
utilization of disk	1	:	.2270	+/- 0.1336E-01
utilization of disk	2	:	.2267	+/- 0.8695E-02

data-base 1		
throughput	:	.2029
mean response time	:	4.645
data-base 2		
throughput	:	.2978
mean response time	:	3.575
administrator		
throughput	:	0.3360E-02
mean response time	:	152.9

```
 /EXEC/ OUTPUT;
... TIME =     12500.00 , NB SAMPLES =     400 , CONF. LEVEL = 0.95
*************************************************************************
*   NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  SERV NB *
*************************************************************************
*           *          *          *          *          *          *
* cpu       * .1899    * .6888    * 1.663    * .4585    *     45341*
*   +/-     *0.2218E-01*0.6434E-01* .4903    * .1257    *          *
*(dbase_1 )* .2031     * .4142    * .5749    * .2819    *     25492*
*(dbase_2 )* .1004     * .1513    * .5916    * .3926    *     18836*
*(admin   )* 1.521     * .1233    * .4969    * 6.128    *      1013*
*           *          *          *          *          *          *
* terminal  * 26.11    * 1.000    * 13.10    * 26.11    *      6259*
*(dbase_1 )* 20.13     * .3122    * 4.089    * 20.13    *      2536*
*(dbase_2 )* 30.18     * .6878    * 9.008    * 30.18    *      3723*
*           *          *          *          *          *          *
* disk   1  * .1212    * .2270    * .2712    * .1448    *     23409*
*   +/-     *0.1853E-02*0.1336E-01*0.1887E-01*0.3563E-02*          *
*(dbase_1 )* .1211     * .2225    * .2652    * .1444    *     22958*
*(admin   )* .1247     *0.4499E-02*0.5959E-02* .1652    *       451*
*           *          *          *          *          *          *
* disk   2  * .1812    * .2267    * .3033    * .2425    *     15634*
*   +/-     *0.4236E-02*0.8695E-02*0.1919E-01*0.1060E-01*          *
*(dbase_2 )* .1812     * .2191    * .2925    * .2419    *     15113*
*(admin   )* .1818     *0.7576E-02*0.1082E-01* .2595    *       521*
*           *          *          *          *          *          *
* source    * 294.9    * 1.000    * 1.000    * 294.9    *        42*
*           *          *          *          *          *          *
* memory    * 4.415    * .8286    * 2.418    * 4.772    *      6298*
*(dbase_1 )* 4.143     * .3110    * .9108    * 4.492    *      2534*
*(dbase_2 )* 2.968     * .3273    * .9916    * 3.329    *      3723*
*(admin   )* 152.6     * .1902    * .5152    * 153.1    *        41*
*           *          *          *          *          *          *
* in_q      * .3566    *0.8478E-01* .1798    * .3566    *      6301*
*(dbase_1 )* .3487     *0.3337E-01*0.7075E-01* .3487    *      2536*
*(dbase_2 )* .3609     *0.5069E-01* .1075    * .3609    *      3723*
*(admin   )* .4574     *0.7248E-03*0.1537E-02* .4574    *        42*
*           *          *          *          *          *          *
* out_q     *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*      6298*
*(dbase_1 )*0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*      2534*
*(dbase_2 )*0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*      3723*
*(admin   )*0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*        41*
*           *          *          *          *          *          *
*************************************************************************
```

The results show that memory contention has a detrimental effect on the performance of the system. The mean response time of dbase_1 and dbase_2 transactions is significantly increased. However, equilibrium conditions are apparently satisfied as indicated by comparing the number of transactions served in stations in_q and out_q.

The mean response time of dbase_1 and dbase_2 transactions can be improved by defining a priority scheduling at memory with dbase_1 and dbase_2 transactions having priority over admin transactions:

```
/STATION/ NAME = memory;
          SCHED = PRIOR;
          PRIOR(dbase_1, dbase_2) = 2;
          PRIOR(admin) = 1;
&
/EXEC/ SIMUL;
```

input parameters

```
mean service time at disk   1  :    .1200
mean service time at disk   2  :    .1800
number of memory partitions:         5.

data-base 1
number of terminals          :         5.
mean service time at cpu     :    .2000
mean thinking time           :   20.00

data-base 2
number of terminals          :        10.
mean service time at cpu     :    .1000
mean thinking time           :   30.00

administrator
mean service time at cpu     :   1.500
mean inter-arrival time      :   250.0
```

results

cpu utilization		: .7417	+/-	0.8611E-01
utilization of disk	1	: .2300	+/-	0.1364E-01
utilization of disk	2	: .2314	+/-	0.1071E-01

data-base 1

throughput	:	.2062
mean response time	:	4.253

data-base 2

throughput	:	.3019
mean response time	:	3.121

administrator

throughput	:	0.3680E-02
mean response time	:	216.3

```
 /EXEC/ OUTPUT;
... TIME =     12500.00 , NB SAMPLES =     400 , CONF. LEVEL = 0.95
*************************************************************************
*   NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  SERV NB *
*************************************************************************
*           *          *          *          *          *          *
* cpu       * .2020    * .7417    * 1.917    * .5223    *      45890*
*   +/-     *0.1959E-01*0.8611E-01* .3835    * .1591    *          *
*(dbase_1 )* .2023    * .4129    * .5568    * .2728    *      25510*
*(dbase_2 )* .1008    * .1522    * .5907    * .3914    *      18868*
*(admin   )* 1.460    * .1766    * .7699    * 6.365    *       1512*
*           *          *          *          *          *          *
* terminal  * 25.60    * 1.000    * 13.05    * 25.60    *       6351*
*(dbase_1 )* 19.84    * .3137    * 4.094    * 19.84    *       2577*
*(dbase_2 )* 29.53    * .6863    * 8.956    * 29.53    *       3774*
*           *          *          *          *          *          *
* disk   1  * .1219    * .2300    * .2730    * .1447    *      23579*
*   +/-     *0.1973E-02*0.1364E-01*0.1845E-01*0.2467E-02*          *
*(dbase_1 )* .1218    * .2235    * .2643    * .1441    *      22934*
*(admin   )* .1260    *0.6501E-02*0.8658E-02* .1678    *        645*
*           *          *          *          *          *          *
* disk   2  * .1817    * .2314    * .3170    * .2490    *      15915*
*   +/-     *0.3346E-02*0.1071E-01*0.2165E-01*0.9525E-02*          *
*(dbase_2 )* .1817    * .2194    * .2995    * .2480    *      15094*
*(admin   )* .1829    *0.1201E-01*0.1755E-01* .2673    *        821*
*           *          *          *          *          *          *
* source    * 267.8    * 1.000    * 1.000    * 267.8    *         46*
*           *          *          *          *          *          *
* memory    * 4.900    * .8556    * 2.753    * 5.379    *       6396*
*(dbase_1 )* 3.983    * .2802    * .9061    * 4.395    *       2576*
*(dbase_2 )* 2.949    * .3038    * 1.044    * 3.459    *       3774*
*(admin   )* 216.3    * .2716    * .8024    * 218.0    *         46*
*           *          *          *          *          *          *
* in_q      * .4796    * .1072    * .2455    * .4796    *       6397*
*(dbase_1 )* .4122    *0.3710E-01*0.8498E-01* .4122    *       2577*
*(dbase_2 )* .5106    *0.6731E-01* .1542    * .5106    *       3774*
*(admin   )* 1.716    *0.2758E-02*0.6316E-02* 1.716    *         46*
*           *          *          *          *          *          *
* out_q     *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*       6396*
*(dbase_1 )*0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*       2576*
*(dbase_2 )*0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*       3774*
*(admin   )*0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*         46*
*           *          *          *          *          *          *
*************************************************************************

 /END/

STOP
```

Confidence intervals estimation

6.1/ Presentation

Simulating a stochastic model consists in generating one or several finite trajectories of the model using random number streams. The required characteristics of the equilibrium (or transient) behavior of the model are then estimated from the information contained in the simulated trajectories. Because of the finite number and length of the generated trajectories the estimates can provide only approximate values of the required characteristics. Therefore, some indications on the accuracy of the estimates are needed.

The standard method of estimating the accuracy of a statistical estimate is to compute confidence intervals. We say that we have a confidence interval for the estimate of a characteristic if we can state that the exact value of the estimated characteristic is within the confidence interval with a specified probability. This probability, expressed in percent, is called the confidence level. In QNAP2, confidence intervals are produced with a confidence level of 95 %. The confidence intervals produced are themselves estimates because the computation of exact confidence intervals would require an a-priori knowledge of the estimated characteristics.

QNAP2 includes three methods for confidence interval estimation:

- the replication method,
- the regeneration method,
- the spectral method.

These methods rely on various assumptions described in the following three sub-sections and have different characteristics in term of computational complexity and memory requirement.

The selection of the confidence interval method and of the parameters specific to each method is made within the /CONTROL/ command with the following parameters:

- ESTIMATION: defines the confidence interval method used (REPLICATION, REGENERATION, SPECTRAL),
- ACCURACY: specifies the list of queues for which confidence intervals are to be produced,

- TEST: introduces a test sequence, i.e. a process which will be activated periodically during the simulation,

- PERIOD: defines the period of activation of the test sequence,

- CORRELATION: defines a list of queues for which autocorrelation functions are to be computed (applicable only to the regeneration method).

In order to illustrate the confidence interval methods we will return to the original model of section 3 and assume that the transactions are issued from a batch worload. The description is as follows:

```
/DECLARE/                & declaration statement

QUEUE
        cpu,             & central processing unit
        batch,           & set of terminals
        disk_1,          & disk unit number 1
        disk_2;          & disk unit number 2

REAL
        t_cpu,           & mean service time at cpu
        t_disk_1,        & mean service time at disk_1
        t_disk_2,        & mean service time at disk_2
        t_batch;         & mean interarrival time of transactions

/STATION/ NAME = cpu;
          SERVICE = EXP(t_cpu);
          TRANSIT = disk_1, 0.3, disk_2, 0.6, OUT;

/STATION/ NAME = disk_1;
          SERVICE = EXP(t_disk_1);
          TRANSIT = cpu;

/STATION/ NAME = disk_2;
          SERVICE = EXP(t_disk_2);
          TRANSIT = cpu;

/STATION/ NAME = batch;
          TYPE = SOURCE;
          SERVICE = EXP(t_batch);
          TRANSIT = cpu;
```

First we will solve the model using the analytical solvers in order to get an exact value of the mean response time of the batch transactions:

```
/CONTROL/ EXIT =
BEGIN
PRINT(" ");
PRINT("mean response time :",(MCUSTNB(cpu)+MCUSTNB(disk_1)+MCUSTNB(disk_2))/
MTHRUPUT(batch));
END;

/EXEC/ BEGIN
       t_cpu:= 0.2;
       t_disk_1:=0.08; t_disk_2:=0.1;
       t_batch:= 5.;
       SOLVE;
       END;
```

```
- CONVOLUTION METHOD ("CONVOL") -
*************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*          *          *          *          *          *          *
* cpu      * .2000    * .4000    * .6667    * .3333    * 2.000    *
*          *          *          *          *          *          *
* batch    * 5.000    * 1.000    * 1.000    * 5.000    * .2000    *
*          *          *          *          *          *          *
* disk_1   *0.8000E-01*0.4800E-01*0.5042E-01*0.8403E-01* .6000    *
*          *          *          *          *          *          *
* disk_2   * .1000    * .1200    * .1364    * .1136    * 1.200    *
*          *          *          *          *          *          *
*************************************************************************
      MEMORY USED:       2806 WORDS OF 4 BYTES
       (  1.08  % OF TOTAL MEMORY)

mean response time :   4.267
```

Then, estimates of the mean response time of transactions and confidence intervals will be produced by simulation using the three confidence intervals methods. In order to do sc, we introduce a resource station system and two additional stations in_q and out_q where the transactions will request and release the resource system. The station system can be considered as a monitoring station used for gathering statistics on the behaviour of transactions in the sub-system cpu-disk_1-disk_2. The introduction of this monitoring station is necessary because confidence intervals are produced only for the standard performance criteria related to stations.

The description of these stations is as follows:

```
/DECLARE/
QUEUE           system,         & monitoring station
        in_q, out_q;    & request and release stations

/STATION/ NAME = system;
          TYPE = RESOURCE, INFINITE;
&
/STATION/ NAME = in_q;
          TYPE = INFINITE;
          SERVICE = P(system);
          TRANSIT = cpu;
&
/STATION/ NAME = out_q;
          TYPE = INFINITE;
          SERVICE = V(system);
          TRANSIT = OUT;
```

Note that the stations system, in_q and out_q are infinite server stations: their function is not to restrict the access of transactions to the sub-system, but to monitor the transactions.

Also, we redefine the routing of transactions in stations batch and cpu and return to the Poisson assumption:

```
/STATION/ NAME = batch;
          SERVICE = EXP(t_batch);
          TRANSIT = in_q;

/STATION/ NAME = cpu;
          TRANSIT = disk_1, 0.3, disk_2, 0.6, out_q;
```

The exit block is also redefined in order to edit the estimated confidence using the confidence intervals access functions.

```
/CONTROL/ EXIT = BEGIN
                 PRINT(" ");
                 PRINT("mean response time: ",
                       MRESPONSE(system)," +/- ", CRESPONSE(system));
                 END;
```

This model will be used in the next sub-sections to illustrate the various confidence interval methods.

6.2/ The replication method

Independent replications is the most straightforward method for obtaining confidence intervals. It consists in generating several sample path of the model studied, so that these sample path are statistically independent and identical. This is achieved by reseting the original initial state of the model at the beginning of each replication, and by using a different random number stream for each replication. In practice a single random number stream is used throughout the whole simulation run so that the random stream for the second replication begins where the stream for the first replication ended, and so on. This guarantees independent random number streams.

Using the /CONTROL/ command the confidence interval method is defined by the parameter ESTIMATION. The list of stations for which the estimation of confidence intervals is requested is specified by the parameter ACCURACY. The value of the parameter ESTIMATION for the regeneration method is REPLICATION. The number of replications is given by the argument of the keyword REPLICATION.

In order to limit the cost of the simulation run a trade-off has to be found between the length of each replication and the number of replications. If the steady-state behavior of the model is looked for, each replication should be long enough to ensure that the steady-state is effectively reached and then it is better to have longer replications and a smaller number of replications. The reverse is true if the simulation is run to analyse the transient behavior of the model. Then shorter replications and a larger number of replications will be preferable.

```
/CONTROL/ TMAX = 2000.;
          ESTIMATION = REPLICATION(5) ;
          ACCURACY = system;
```

We can now request the simulation of the model:

```
/EXEC/ SIMUL;


*** SIMULATION WITH REPLICATION METHOD ***
... MEAN SIMULATION TIME =     2000.000
    NUMBER OF REPLICATIONS =       5
    CONFIDENCE LEVEL = 0.95
==>WARNING (OUTPUT) : LESS THAN 10 REPLICATIONS
                CONFIDENCE INTERVALS MAY NOT BE MEANINGFUL
*****************************************************************
*   NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  SERV NB *
*****************************************************************
*           *          *          *          *          *          *
* cpu       * .1986    * .3811    * .6302    * .3283    * 3838.    *
*           *          *          *          *          *          *
* batch     * 5.101    * 1.000    * 1.000    * 5.101    * 391.2    *
*           *          *          *          *          *          *
* disk_1    *0.8139E-01*0.4778E-01*0.5032E-01*0.8572E-01* 1173.    *
*           *          *          *          *          *          *
* disk_2    * .1007    * .1145    * .1289    * .1133    * 2274.    *
*           *          *          *          *          *          *
* system    * 4.136    * .4772    * .8094    * 4.136    * 390.6    *
*   +/-     * .2416    *0.3292E-01*0.7185E-01* .2416    *          *
*           *          *          *          *          *          *
* in_q      *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00* 391.2    *
*           *          *          *          *          *          *
* out_q     *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00* 390.6    *
*           *          *          *          *          *          *
*****************************************************************
... END OF SIMULATION ...


      MEMORY USED:       3622 WORDS OF 4 BYTES
       (  1.39  % OF TOTAL MEMORY)

mean response time:    4.136     +/-     .2416
```

The confidence interval obtained for the mean response time of the transactions is (3.8944 , 4.3776). This interval contains the exact value 4.267.

6.3/ The regeneration method

The basic principle is to split the simulation run into several successive intervals, so that the model behavior in these intervals be statistically equivalent and independent. Thus there are as many independent sub-simulations as there are intervals and the results within each simulation sub-run are

considered as samples of independent identically distributed random variables; these variables will be used to estimate the confidence intervals and the final results.

In practice, the independence hypothesis may be satisfied if the model corresponds to a stochastic regenerative process. In this case the model contains one or more special states, called regeneration states. Whenever the system returns to one of these states a regeneration point is obtained, i.e. a point where the past behavior of the system has no more influence on its future behavior. The regeneration points may be used to split the simulation duration into intervals which satisfy the statistical independence hypothesis.

The determination of exact regeneration intervals is not, in general, feasible on reasonably complex models. The simplest case is the case of Markovian models in which every state is a regeneration state. Similarly, in an open network model having a Poisson arrival process the state "all the stations empty" is generally a regeneration state.

The user may define regeneration points by testing a condition defining the regeneration states and by explicitly requesting the computation of partial statistics on the last interval if the test is satisfied. The more general way to test a condition is to program the test in a special process, called a test sequence, which is activated periodically with a period specified by the user with the parameter PERIOD. If a zero period is explicitly specified the test sequence is executed whenever an event is processed by the simulator: the condition is then "continuously" tested. A test sequence is defined within the /CONTROL/ command by the parameter TEST.

Whenever the specified regeneration state is reached, the computation of the partial statistics on the last interval must be requested explicitly using the QNAP2 procedure SAMPLE. The function of SAMPLE is the computationn of partial statistics on the last regeneration interval and of global statistics and confidence intervals using the information collected from the beginning of the simulation up to the current regeneration point. In this way partial results on the simulation are available before the simulation run is completed. However, these intermediate results are not output unless explicitly requested by a call to the procedure OUTPUT.

In the previous example, the state "system empty" is a regeneration state because the arrival process is Poisson. The condition "system empty" can be coded in the test sequence using the predefined queue attribute NB, and if this condition is satisfied the procedure SAMPLE is activated.

Using the /CONTROL/ command we now specify the regeneration method with the parameter ESTIMATION = REGENERATION and specify the test sequence and activ period.

```
/CONTROL/ ESTIMATION = REGENERATION;
          TMAX = 10000.;
          PERIOD = 0.; TEST = IF system.NB = 0 THEN SAMPLE;
&
/EXEC/ SIMUL;
```

```
***SIMULATION WITH REGENERATIVE METHOD ***
... TIME =     9999.01 , NB SAMPLES =   1017 , CONF. LEVEL = 0.95
*************************************************************************
*   NAME   *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  SERV NB *
*************************************************************************
*          *          *          *          *          *          *
* cpu      * .1986    * .3731    * .6108    * .3252    *      18781*
*          *          *          *          *          *          *
* batch    * 5.197    * 1.000    * 1.000    * 5.197    *       1923*
*          *          *          *          *          *          *
* disk_1   *0.8208E-01*0.4669E-01*0.4917E-01*0.8644E-01*       5688*
*          *          *          *          *          *          *
* disk_2   * .1007    * .1126    * .1266    * .1133    *      11171*
*          *          *          *          *          *          *
* system   * 4.087    * .4684    * .7866    * 4.087    *       1921*
*   +/-    * .3618    *0.2783E-01*0.8810E-01* .3618    *          *
*          *          *          *          *          *          *
* in_q     *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*       1923*
*          *          *          *          *          *          *
* out_q    *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*       1921*
*          *          *          *          *          *          *
*************************************************************************
... END OF SIMULATION ...

     MEMORY USED:       3620 WORDS OF 4 BYTES
      (  1.39  % OF TOTAL MEMORY)

mean response time:    4.087     +/-     .3618
```

The confidence interval obtained is (3.725 , 4.448). The theoretical value is 4.267.

Another way to test a regeneration state is to program this test in a service, provided that the condition is met only when this service is active. This is in general a more efficient approach because the condition will be tested less frequently than in the previous case.

In the example, the condition can be tested in the out_q:

```
/STATION/ NAME = out_q;
          SERVICE = BEGIN
                    V(system);
                    IF system.NB = 0 THEN SAMPLE;
                    END;
```

The periodic activation of the TEST sequence is deactivated by:

```
/CONTROL/ PERIOD = ;

/EXEC/ SIMUL;
```

```
***SIMULATION WITH REGENERATIVE METHOD ***
... TIME =    10000.00 , NB SAMPLES =   1017 , CONF. LEVEL = 0.95
*************************************************************************
*   NAME   *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  SERV NB *
*************************************************************************
*          *          *          *          *          *          *
* cpu      * .1986    * .3732    * .6109    * .3252    *     18786*
*          *          *          *          *          *          *
* batch    * 5.197    * 1.000    * 1.000    * 5.197    *      1923*
*          *          *          *          *          *          *
* disk_1   *0.8207E-01*0.4669E-01*0.4917E-01*0.8643E-01*      5689*
*          *          *          *          *          *          *
* disk_2   * .1007    * .1126    * .1266    * .1133    *     11176*
*          *          *          *          *          *          *
* system   * 4.087    * .4685    * .7867    * 4.087    *      1921*
*   +/-    * .3618    *0.2783E-01*0.8809E-01* .3618    *          *
*          *          *          *          *          *          *
* in_q     *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*      1923*
*          *          *          *          *          *          *
* out_q    *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*      1921*
*          *          *          *          *          *          *
*************************************************************************
... END OF SIMULATION ...


     MEMORY USED:       3639 WORDS OF 4 BYTES
      (  1.39  % OF TOTAL MEMORY)

mean response time:    4.087     +/-     .3618
```

The results obtained (point estimates and confidence intervals) are identical to the previous ones. Indeed the two simulation runs use the same random number stream, have the same length and the same regeneration points are used. The differnces between the two simulations lies in the way the regeneration point are detected.

In many situations approximate regeneration points for a given model can be derived from the existence of exact regeneration states for a simpler version of the model studied. For example, in the previous model the existence of a regeneration state is related to the Poisson assumption on the arrival process. If the arrival process is not Poisson, the state "system empty" is no longer a regeneration point. However, it may still be considered as an approximate regeneration point as in the next example where hyper-exponential inter-arrival times are assumed:

```
/STATION/ NAME = batch;
          SERVICE = HEXP(t_batch,5.);
```

However, one may question the validity of the confidence intervals which will be obtained because of the approximate regeneration points used. Indeed, the statistical independence of the intervals must be thoroughly assessed before meaningfull interpretations of the confidence intervals are made. For this purpose one can request the computation of the autocorrelation functions on the basic quantities used in the derivation of the standard performance criteria. This is done with the parameter CORRELATION of the /CONTROL/ command. The value of the parameter CORREL is the list of queues for which autocorrelation functions are requested. The maximum order of the autocorrelation functions is defined by an integer following the list of queues (the default value for the order is 5).

```
/CONTROL/ CORREL = system, 5;
```

```
/EXEC/ SIMUL;


***SIMULATION WITH REGENERATIVE METHOD ***
... TIME =     10000.00 , NB SAMPLES =     627 , CONF. LEVEL = 0.95
*************************************************************************
*   NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  SERV NB *
*************************************************************************
*           *          *          *          *          *          *
* cpu       * .2035    * .4064    * .9653    * .4834    *      19969*
*           *          *          *          *          *          *
* batch     * 4.859    * 1.000    * 1.000    * 4.859    *       2042*
*           *          *          *          *          *          *
* disk_1    *0.8195E-01*0.4923E-01*0.5320E-01*0.8856E-01*       6007*
*           *          *          *          *          *          *
* disk_2    *0.9986E-01* .1190    * .1449    * .1216    *      11920*
*           *          *          *          *          *          *
* system    * 5.698    * .4705    * 1.163    * 5.698    *       2042*
*   +/-     * .5225    *0.4590E-01* .1624    * .5225    *          *
*           *          *          *          *          *          *
* in_q      *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*       2042*
*           *          *          *          *          *          *
* out_q     *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*       2042*
*           *          *          *          *          *          *
*************************************************************************
```

AUTOCORRELATION FUNCTIONS ON QUEUE MEASURES

ORDER	BLOC SIZE	SERV TIME	BUSY TIME	QUEUE LENGTH	RES TIME	NB SERVED	BLOCKED TIME
AUTOCORRELATION ON QUEUE system							
1	0.083	0.016	0.052	0.016	0.016	0.043	0.016
2	-0.012	0.027	0.026	0.027	0.027	0.005	0.027
3	0.002	-0.020	-0.015	-0.020	-0.020	-0.006	-0.020
4	0.014	0.020	0.030	0.020	0.020	0.026	0.020
5	0.032	-0.022	0.007	-0.022	-0.022	0.037	-0.022

END OF AUTOCORRELATION COMPUTATIONS
... END OF SIMULATION ...

MEMORY USED: 3839 WORDS OF 4 BYTES
(1.47 % OF TOTAL MEMORY)

mean response time: 5.698 +/- .5225

The values of the autocorrelation coefficients confirm the validity of the state "system empty" as a regeneration point. However, it can be observed that the confidence interval obtained is significantly larger than with the previous model: because of the hyper-exponential inter-arrival process the variance of the response time is greater.

Without theoretical results determining the regeneration states of the model, one will choose artificial regeneration points. For example, spliting the simulation run into fixed length sub-runs produces intervals which satisfy the independence hypothesis if the length of the intervals is large enough. The end of each interval can then be considered as an approximate regeneration point.

This method can be simply implemented by causing the periodic activation of the procedure SAMPLE as shown in the following example:

```
/CONTROL/ TEST =SAMPLE; PERIOD = 100.;
```

We also have to deactivate the call to the procedure SAMPLE in the service description of the station out_q:

```
/STATION/ NAME = out_q;
          SERVICE = V(system);
```

The length of the intervals should be large enough so that within an interval the behavior of the model be as independent as possible of its behavior in the previous interval. On the other hand, the number of intervals should be large enough to ensure an accurate estimation of the confidence intervals. A useful rule of thumb is to have 100 fixed intervals.

After the Poisson assumption is resumed the analysis is requested:

```
/STATION/ NAME = batch;
          SERVICE = EXP(t_batch);

/EXEC/ SIMUL;
```

```
***SIMULATION WITH REGENERATIVE METHOD ***
... TIME =     10000.00 , NB SAMPLES =     100 , CONF. LEVEL = 0.95
*************************************************************************
*   NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  SERV NB *
*************************************************************************
*           *          *          *          *          *          *
* cpu       * .1986    * .3732    * .6109    * .3252    *     18786*
*           *          *          *          *          *          *
* batch     * 5.197    * 1.000    * 1.000    * 5.197    *      1923*
*           *          *          *          *          *          *
* disk_1    *0.8207E-01*0.4669E-01*0.4917E-01*0.8643E-01*      5689*
*           *          *          *          *          *          *
* disk_2    * .1007    * .1126    * .1266    * .1133    *     11176*
*           *          *          *          *          *          *
* system    * 4.087    * .4685    * .7867    * 4.087    *      1921*
*   +/-     * .3897    *0.2779E-01*0.8497E-01* .3897    *          *
*           *          *          *          *          *          *
* in_q      *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*      1923*
*           *          *          *          *          *          *
* out_q     *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*      1921*
*           *          *          *          *          *          *
*************************************************************************
```

AUTOCORRELATION FUNCTIONS ON QUEUE MEASURES

AUTOCORRELATION ON QUEUE system							
ORDER	BLOC SIZE	SERV TIME	BUSY TIME	QUEUE LENGTH	RES TIME	NB SERVED	BLOCKED TIME
1	0.000	0.086	0.091	0.175	0.086	0.161	0.086
2	0.000	-0.009	-0.033	0.012	-0.009	0.054	-0.009
3	0.000	0.048	0.056	0.049	0.048	0.096	0.048
4	0.000	-0.040	0.084	0.023	-0.040	0.060	-0.040
5	0.000	-0.071	-0.140	-0.095	-0.071	-0.055	-0.071

END OF AUTOCORRELATION COMPUTATIONS
... END OF SIMULATION ...

MEMORY USED: 3854 WORDS OF 4 BYTES
(1.48 % OF TOTAL MEMORY)

mean response time: 4.087 +/- .3897

The results show that the independence assumption is satisfied with this model with intervals of length 100.;

In conclusion it is important to recall that the independence assumption between the consecutive intervals created by the regeneration points is essential for the correct estimation of confidence intervals. Meaningless confidence intervals would be produced if the independence is not verified.

6.4/ The spectral method

The basic principle of the two previous methods was to build a set of independent and identically distributed samples and to apply standard statistical techniques for estimating confidence intervals. The spectral method is somewhat diferent in the sense that it does not rely on the independence assumption and applies to correlated samples provided that the identical distribution property is satisfied. Indeed, the correlation is explicitly taken into account in the estimation of the confidence intervals. As a consequence, the confidence intervals produced by the spectral method will be most often larger than those produced by the regenerative method because of the effect of correlation in the variance of the point estimates.

The main advantage of the spectral method over the two previous methods is that the user needs not bother with the choice of specific parameters (number of replications, regeneration state,..): the method applies to the analysis of the stationary behaviour of any simulated model.

The value of the parameter ESTIMATION for the spectral method is ESTIMATION = SPECTRAL. The value of the first measurement interval of the spectral method can be defined by the argument of the keyword SPECTRAL. This argument is optional and the default value TMAX/400. will be used if no value is given.

First we cancel the previous control parameters specific to the regeneration method and then request the spectral method (note that a call to the SAMPLE procedure in a simulation run while the spectral method is active would cause an error).

```
/CONTROL/ CORREL = NIL;
          PERIOD = ;

          ESTIMATION = SPECTRAL;
```

The analysis is now requested:

```
 /EXEC/ SIMUL;


***SIMULATION WITH SPECTRAL METHOD ***
... TIME =     10000.00 , NB SAMPLES =     400 , CONF. LEVEL = 0.95
*************************************************************************
*   NAME   *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  SERV NB *
*************************************************************************
*          *          *          *          *          *          *
* cpu      * .1986    * .3732    * .6109    * .3252    *     18786*
*          *          *          *          *          *          *
* batch    * 5.197    * 1.000    * 1.000    * 5.197    *      1923*
*          *          *          *          *          *          *
* disk_1   *0.8207E-01*0.4669E-01*0.4917E-01*0.8643E-01*      5689*
*          *          *          *          *          *          *
* disk_2   * .1007    * .1126    * .1266    * .1133    *     11176*
*          *          *          *          *          *          *
* system   * 4.087    * .4685    * .7867    * 4.087    *      1921*
*   +/-    * .4850    *0.3597E-01* .1209    * .4850    *          *
*          *          *          *          *          *          *
* in_q     *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*      1923*
*          *          *          *          *          *          *
* out_q    *0.0000E+00*0.0000E+00*0.0000E+00*0.0000E+00*      1921*
*          *          *          *          *          *          *
*************************************************************************
... END OF SIMULATION ...


      MEMORY USED:       4810 WORDS OF 4 BYTES
       (  1.84  % OF TOTAL MEMORY)

mean response time:    4.087     +/-     .4850
```

The results show that the confidence intervals obtained with the spectral method are somewhat larger than with the regeneration approach. This was to be expected because no assumption on the independence of the samples is made and as a consequence covariance terms are included in the estimate of the variance.

```
 /END/

STOP
```

Analysis of simple product-form networks

7.1/ Presentation

This example illustrates how the various analytical solvers of QNAP2 can be used depending on the assumptions of the model considered. The Markovian solver is used concurrently on the same model in order to check the accuracy of the results produced by the approximate analytical solvers.

The model consists of four stations (terminal, cpu, disk1 and disk2) with two classes of customers (c1 and c2). The services of the two classes of customers at stations terminal and cpu are distinct. In a first step it is assumed that the station terminal is infinite and that the station cpu is processor-sharing, so that the product-form assumptions are satisfied.

7.2/ QNAP2 description of the model

The QNAP2 description of this simple model is as follows:

```
/DECLARE/ QUEUE terminal, cpu, disk1, disk2;
          CLASS c1, c2;
          INTEGER i;

/STATION/ NAME = terminal; TYPE = INFINITE;
          INIT(c1) = 2; INIT(c2) = 1;
          SERVICE(c1) = EXP(1000); SERVICE(c2) = EXP(2000);
          TRANSIT = cpu;

/STATION/ NAME = cpu; SCHED = PS;
          SERVICE(c1) = EXP(10);
          TRANSIT(c1) = disk1,10,disk2,15,terminal,1;
          SERVICE(c2) = EXP(20);
          TRANSIT(c2) = disk1,15,disk2,5,terminal1,1;

/STATION/ NAME = disk1;
          SERVICE = EXP(30); TRANSIT = cpu;

/STATION/ NAME = disk2;
          SERVICE = EXP(25); TRANSIT = cpu;
```

7.3/ Description of the analysis options

The model is analysed using the analytical solvers (SOLVE procedure) and then the Markovian solver (MARKOV procedure). Only the throughput at station terminal for both classes is edited. The calls to the solvers and the edition of the results is specified in a macro-statement which will be used for all the next steps of the example.

```
/CONTROL/ OPTION = NRESULT;
          CLASS = ALL QUEUE;

$MACRO resolution

/EXEC/ BEGIN
       PRINT(" ");
       PRINT("solver     c1          c2");
       SOLVE;
       PRINT("SOLVE :",MTHRUPUT(terminal, c1), MTHRUPUT(terminal, c2));
       MARKOV;
       PRINT("MARKOV:",MTHRUPUT(terminal, c1), MTHRUPUT(terminal, c2));
       END;

$END
```

7.4/ Analysis of various model configurations

The analyses are now requested:

```
$resolution

solver      c1           c2
SOLVE :  0.9080E-03  0.3015E-03
MARKOV:  0.9080E-03  0.3016E-03
```

The dispatching algorithm selects the Mean Value Analysis algorithm. As would be shown in the standard report the Markovian solver identifies 50 states for this model. It is verified that similar results are produced by the two solvers.

The second step consists in modifying the service distributions at the station cpu and checking that still similar results are obtained. However, the Markovian solver identifies in this case a greater number of states (92) because of the internal Coxian representation of the non-exponential distributions.

```
/STATION/ NAME = cpu; SCHED = PS;
          SERVICE(c1) = HEXP(10,4); SERVICE(c2) = ERLANG(20,2);

$resolution

solver      c1            c2
SOLVE :  0.9080E-03  0.3015E-03
MARKOV:  0.9079E-03  0.3015E-03
```

We consider now a priority, preemptive scheduling at station cpu, assuming that the class c2 has the highest priority. The model is no longer a product form network. The dispatching algorithm selects then the approximate solver PRIORPR. The Markovian solver provides a reference for checking the accuracy of the approximation.

```
/STATION/ NAME = cpu;
          SERVICE(c1) = EXP(10); SERVICE(c2) = EXP(20);
          SCHED = PRIOR, PREEMPT;
          PRIOR(c1) = 1; PRIOR(c2) = 2;

$resolution

solver      c1            c2
SOLVE :  0.8851E-03  0.3138E-03
MARKOV:  0.8712E-03  0.3192E-03
```

Coming back to the initial assumptions of step 1, the topology of the network is changed so that a customer visits successively each station. The mean service times are also changed so that the global requests of a customer remain unchanged.

```
/STATION/ NAME = cpu;
          SERVICE(c1) = EXP(26*10);
          SERVICE(c2) = EXP(21*20);
          TRANSIT(c1,c2) = disk1;
          SCHED = PS;

/STATION/ NAME = disk1;
          SCHED  =  PS;
          SERVICE(c1) = EXP(30*10); SERVICE(c2) = EXP(30*15);
          TRANSIT(c1,c2) = disk2;

/STATION/ NAME = disk2;
          SCHED  =  PS;
          SERVICE(c1) = EXP(15*25); SERVICE(c2) = EXP(5*25);
          TRANSIT(c1,c2) = terminal;

$resolution

solver      c1            c2
SOLVE :  0.9080E-03  0.3015E-03
MARKOV:  0.9080E-03  0.3015E-03
```

The results are similar to those obtained in steps 1 and 2 due to the fact that the steady-state solution of a product network is independent of its topology.

This property is no longer verified for non product-form networks as shown in the next step. If priority preemptive scheduling is again assumed at station cpu as in step 3, the exact results are not similar to those obtained previously.

```
/STATION/ NAME = cpu; SCHED = PRIOR, PREEMPT;

$resolution

solver      c1            c2
SOLVE :  0.8885E-03  0.3137E-03
MARKOV:  0.8795E-03  0.3130E-03


/END/

STOP
```

A Markovian model of a multi-processor architecture

8.1/ Presentation

This example shows the application of the Markovian solver for the analysis of a computer architecture involving few customers and complex synchronization schemes.

The model represents in a simplified way a multi-microprocessor system with shared memories. An asynchronous bus is the global communication support between the processors. Each processor is linked to its own memory through a local bus. A processor can access any non-local memory through the global bus.

In order to access a non local-memory a processor must first request the global bus. Then a bus delay, t_bus, is incurred before the memory access can be performed. If the memory is busy serving a local request, the non-local request waits until the completion of the local access. The global bus is busy until the non-local access is completed.

This system is modeled as a queueing network with resource possession. It can be analysed using discrete event simulation, approximate analysis °§, or Markovian analysis as presented here.

The following parameters are used to describe the behavior of a processor:

- t_proc: mean active processor time between two consecutive memory accesses (local or non-local);
- p_local: probability of a local memory access;
- t_local: mean memory service time for local accesses;
- t_global: mean memory service time for non-local accesses.

A non-local access is distributed with equal probabilities over the non-local memories.

8.2/ QNAP2 description of the model

The QNAP2 description of the model is parametrized by the number, n, of processors and memories. A user-defined attribute used as an index is added to the object types QUEUE and CLASS. With each processor proc(i) is associated a local memory mem(i) and the activity of processor proc(i) is represented by a customer of class c(i).

```
/DECLARE/ QUEUE INTEGER ip;  & index of a queue
          CLASS INTEGER ic;  & index of a class

          INTEGER n = 3;        & number of processors

          QUEUE proc(n), mem(n);
          QUEUE bus, wait_bus;

          CLASS c(n);
          INTEGER i;

          REAL t_proc,       & mean time between memory accesses
               t_local,      & mean memory access time (local)
               t_global,     & mean memory access time (global)
               p_local,      & proportion of local accesses
               t_bus;        & mean bus time (excluding memory access)
          BOOLEAN bus_free;
```

The description of the stations proc, mem and bus makes use of the queue and class attributes ip and ic. This allows a concise expression of the routing rules for any configuration of the model.

```
/STATION/ NAME = proc;
          INIT = IF ip = ic THEN 1 ELSE 0;
          SERVICE = EXP(t_proc);
          TRANSIT = mem(ic), p_local, wait_bus;

/STATION/ NAME = mem;
          SERVICE = IF ic = ip THEN EXP(t_local)
                               ELSE EXP(t_global);
          TRANSIT = proc(ic);

/STATION/ NAME = bus;
          SERVICE = EXP(t_bus);
          TRANSIT = mem WITH QUEUE<>mem(ic), 1.0 REPEAT n-1;
```

8.3/ Specification of the synchronizations

In the present version of QNAP2, the synchronisation procedures P and V can only be used with simulation. Thus, the synchronisations have to be specified by describing the transition rules. In the model, a customer requesting a non-local memory is sent to a simple FIFO queue, wait_bus. The bus is allocated to the first customer of this queue as soon as the bus is freed by the previous non-local request. This condition is expresssed in a test sequence, activated at each state transition of the model. The current state of the model is

accessed through the predefined function CUSTNB which return the current number of customers in a given queue. The condition "bus available" is tested by checking in all the memories that there is no non-local access waiting or being processed.

```
/CONTROL/ TEST =
          BEGIN
          bus_free :=  TRUE;
          FOR i := 1 STEP 1 UNTIL n DO
             IF ( (CUSTNB(mem(i)) = 1) AND (CUSTNB(mem(i),c(i)) = 0) )
                                       OR  (CUSTNB(mem(i))>1)
             THEN bus_free := FALSE;

          IF bus_free THEN MOVE(wait_bus, bus);
          END;

          OPTION = NRESULT;
```

8.4/ Specification of the analysis options and results

In the next /EXEC/ command the indices of the processors, memories and classes are initialized. Then the parameters of the models are set and the analysis is requested for several values of the local access probability. The edited performance criteria is the efficiency of the system defined as the ratio between the effective throughput of a processor (in terms of the number of memory accesses per unit of time) and the throughput which would be obtained if no bus and memory contention occured.

```
/EXEC/ BEGIN
       FOR i := 1 STEP 1 UNTIL n DO
         BEGIN proc(i).ip := i; c(i).ic := i; mem(i).ip := i; END;

       t_proc := 10; t_local := 2; t_global := 3;
       t_bus := 4;

       PRINT(" ");
       FOR p_local :=  0.8, 0.4, 0.2 DO
            BEGIN
            MARKOV;

            PRINT("Percentage of local access = ",p_local,
                  "Efficiency = ",MTHRUPUT(proc(1))*(t_proc+p_local*t_local+
                                                     (1-p_local)*(t_bus+t_global)) );
            END;

     END;
```

```
Percentage of local access =    .8000    Efficiency =    .9752
Percentage of local access =    .4000    Efficiency =    .8990
Percentage of local access =    .2000    Efficiency =    .8574
```

The example analysed here (3 processors) lead to a Markov chain with 274 states and 1051 non-zero elements in the transition matrix.

STOP

Hierarchical and hybrid modelling

9.1/ Presentation

This example shows how a hierarchical and hybrid mdelling approach can be conducted with QNAP2. The system considered is a transactional sytem consisting of:

- a central processing unit;
- a set of disk sub-systems;
- a set of remote terminals accessing the system through a network;
- a set of local terminals.

Each disk sub-system is composed of the number of moving head disks and of a block multiplexor channel shared by the disks. The channel is allocated to the disk i/o requests according to a FIFO policy. When allocated to a disk request it is used throughout the search and transfer phases of the i/o operation. It is assumed that within a disk sub-system the disks are accessed with equal probabilities.

The performance of the system are analysed by a two level modelling approach. At the first level, the disk sub-systems are considered in order to estimate the effect of channel contention and its output rate as a function of the current number of i/o requests in a disk sub-system. The second modelling levels deals with the whole system where the disk sub-systems are represented by equivalent stations having as service rate the output rate obtained at the first modelling level.

9.2/ Description of the model

9.2.1/ Description of the whole system

The model of the whole system follows standard modelling techniques. Two classes of transactions are considered: local transactions (local) and incoming remote transactions (remote). An additional class is considered to describe the routing rules of returning remote transactions (return) through the network. The disk sub-systems are represented by stations with state dependent service rates (array io_rate) and exponential service times. The sets of terminals are described as an infinite servers station and the network is represented by a delay station with state dependent service rate (array net_rate).

```
/DECLARE/ QUEUE cpu, io(5), terminal, network;

          CLASS REAL t_cpu, visit(5), think;
          CLASS INTEGER users;

          CLASS remote, local, return;
          INTEGER n_io;

          REAL net_rate(25)=1. REPEAT 25,
               io_rate(25);
```

The variables defining the behavior of each class of transactions are defined as attributes of the object type CLASS: t_cpu (mean cpu time), visit(i) (visit ratio to the i-th i/o system), think (mean think time), users (number of active users). The number of actual i/o systems is defined by the variable n_io.

```
/STATION/ NAME = terminal;
          TYPE = INFINITE;
          SERVICE(remote, local) = EXP(think);
          INIT(remote, local) = users;
          TRANSIT(local) = cpu;
          TRANSIT(remote) = network;

/STATION/ NAME = cpu;
          SCHED = PS;
          SERVICE(remote, local) = EXP(t_cpu);
          TRANSIT(remote) = io(1 STEP 1 UNTIL n_io),
                            visit(1 STEP 1 UNTIL n_io), network, return, 1;
          TRANSIT(local) = io(1 STEP 1 UNTIL n_io),
                           visit(1 STEP 1 UNTIL n_io), terminal, 1;

/STATION/ NAME = network;
          TYPE = INFINITE;
          SERVICE = EXP(0.005);
          RATE = net_rate;
          TRANSIT(return) = terminal, remote;
          TRANSIT(remote) = cpu;

/STATION/ NAME = io;
          SERVICE = EXP(1.);
          RATE = io_rate;
          TRANSIT = cpu;
```

9.2.2/ The disk sub-systems

The description of a disk sub-system is straightforward. An array of queues is used to represent the queues associated with the disks. The actual number of disks in an i/o subsystem is defined by the variable n_disk. The channel is defined as a passive resource requested and released by the i/o requests. The sequence of phases involved in an i/o operation is described in the service of each disk station. The performance figures of the disks are given in milli-seconds.

```
/DECLARE/ QUEUE disk(10), channel;
          REAL seek=30., transfer=4., rotation=16.;
          INTEGER request, n_disk;
          CLASS i_o;

/STATION/ NAME = disk;
          SERVICE = BEGIN
                    EXP(seek);
                    P(channel);
                    UNIFORM(0., rotation);
                    CST(transfer);
                    V(channel);
                    END;
          TRANSIT = disk(1 STEP 1 UNTIL n_disk), (1 REPEAT n_disk);

/STATION/ NAME = channel;
          TYPE = RESOURCE;

/STATION/ NAME = disk(1);
          INIT(i_o) = request;
```

9.3/ Analysis of the model

We now specify the different steps of the analysis of the model:

- sub-systems analysis;
- whole system analysis.

For sake of simplicity, it is assumed that the number of disks in a subsystem and the performance parameters of the disks remain the same throughout a session. As a consequence, the sub-system analysis is performed only once. Then different configurations of the whole system are investigated.

The specification of the analysis is made within one /EXEC/ command using the algorithmic language level of QNAP2.

9.3.1/ Sub-system analysis

First, the number of disks in a sub-system is entered and then the sub-system is analysed with an increasing number of i/o requests until no significant change in the output rate is observed. The output rate of a sub-system with i requests is stored in io_rate(i). The procedure call NETWORK( ) restricts the analysis to the network made of the station channel and the first n_disk disks. The analysis is performed by simulation (call to the procedure SIMUL) because of the synchronization operations involved.

```
/DECLARE/ REF CLASS r_class;
          OBJECT answer; END;
          answer yes, no;
          REAL tmax, x;
          INTEGER i, i0;
          LABEL l1, l2;

/CONTROL/ CLASS = ALL QUEUE; TMAX = tmax;
          OPTION = NRESULT;

/EXEC/ BEGIN
       PRINT("number of disks per i/o system?");
       n_disk:= GET(INTEGER);
       PRINT(" ");
       NETWORK(channel, disk(1 STEP 1 UNTIL n_disk));
       request:= 0;
   l1:request:= request+1;
       tmax:= 10000;
       SIMUL;
       io_rate(request):= MTHRUPUT(channel)*1000;
       PRINT("i/o requests:", request," throughput:",io_rate(request),
            " i/o per second");
       IF request=25 THEN GOTO l2;
       IF request=1 THEN GOTO l1;
       IF (io_rate(request)-io_rate(request-1))/io_rate(request) > 0.1
                                  THEN GOTO l1;
       i0:=request;
       FOR i:= i0 STEP 1 UNTIL 25 DO io_rate(i):= io_rate(i0);
```

9.3.2/ Global system analysis

The analysis of the global system performance is now specified. The number of i/o subsystems and the characteristics of the remote and local workloads are entered and the analysis of the model by analytical methods is requested (call to the procedure SOLVE).

```
12:PRINT(" ");
   PRINT("number of i/o systems?");
   n_io:= GET(INTEGER);
   FOR r_class:= remote, local DO
   BEGIN
   PRINT(" ");
   PRINT(r_class," users:");
   PRINT(" ");
   PRINT("number of users ?");
   r_class.users:= GET(INTEGER);
   PRINT("mean think time?");
   r_class.think:= GET(REAL);
   PRINT("mean cpu time?");
   r_class.t_cpu:= GET(REAL);
   PRINT("visit ratios to i/o systems?");
   x:=0.0;
   FOR i:= 1 STEP 1 UNTIL n_io DO BEGIN
                                  r_class.visit(i):= GET(REAL);
                                  x:= x+r_class.visit(i);
                                  END;
   r_class.t_cpu:= r_class.t_cpu/x;

   END;
   NETWORK(terminal, network, cpu, io(1 STEP 1 UNTIL n_io));
   SOLVE;
   PRINT(" ");
   FOR r_class:= local, remote DO
         PRINT(r_class," users mean response time:",
         r_class.users/MTHRUPUT(terminal, r_class)-r_class.think);
   PRINT(" ");
   PRINT("complete results (yes/no)?");
   IF GET(answer)=yes THEN OUTPUT;
   PRINT(" ");
   PRINT("new configuration (yes/no)?");
   IF GET(answer)=yes THEN GOTO 12;
      END;
```

9.4/ Numerical example

We consider that the disk sub-systems are comprized of three disks. Then a configuration with three sub-systems is analysed:

```
number of disks per i/o system?
3

i/o requests:        1. throughput:   23.40      i/o per second
```

```
i/o requests:        2. throughput:   39.20     i/o per second
i/o requests:        3. throughput:   44.50     i/o per second
i/o requests:        4. throughput:   48.10     i/o per second

number of i/o systems?
3

remote     users:

number of users ?
5
mean think time?
30
mean cpu time?
0.6
visit ratios to i/o systems?
8 3 6

local      users:

number of users ?
10
mean think time?
25
mean cpu time?
1.2
visit ratios to i/o systems?
3 10 6

local      users mean response time:   3.235
remote     users mean response time:   2.010

complete results (yes/no)?
yes
```

NAME	SERVICE	BUSY PCT	CUST NB	RESPONSE	THRUPUT
cpu	*0.5524E-01	.5466	1.046	,1057	9.895
(remote	)*0.3529E-01	*0.9923E-01	.1961	*0.6976E-01	2.812
(local	)*0.6316E-01	.4474	.8499	.1200	7.083
io	1 *0.4120E-01	*0.9526E-01	.1007	*0.4354E-01	2.312
(remote	)*0.4125E-01	*0.5154E-01	*0.5437E-01	*0.4351E-01	1.250
(local	)*0.4115E-01	*0.4372E-01	*0.4630E-01	*0.4357E-01	1.063
io	2 *0.4012E-01	.1609	.1773	*0.4420E-01	4.010
(remote	)*0.3997E-01	*0.1873E-01	*0.2076E-01	*0.4430E-01	.4686
(local	)*0.4014E-01	.1422	.1565	*0.4419E-01	3.542

```
* io     3 *0.4069E-01* .1246    * .1343    *0.4384E-01* 3.062    *
*(remote  )*0.4068E-01*0.3813E-01*0.4109E-01*0.4385E-01* .9372    *
*(local   )*0.4070E-01*0.8648E-01*0.9316E-01*0.4384E-01* 2.125    *
*          *          *          *          *          *          *
* terminal * 26.53    *0.0000E+00* 13.54    * 26.53    * .5104    *
*(remote  )* 30.00    *0.0000E+00* 4.686    * 30.00    * .1562    *
*(local   )* 25.00    *0.0000E+00* 8.854    * 25.00    * .3542    *
*          *          *          *          *          *          *
* network  *0.5000E-02*0.0000E+00*0.1562E-02*0.5000E-02* .3124    *
*(remote  )*0.5000E-02*0.0000E+00*0.7810E-03*0.5000E-02* .1562    *
*(return  )*0.5000E-02*0.0000E+00*0.7810E-03*0.5000E-02* .1562    *
*          *          *          *          *          *          *
*************************************************************************

new configuration (yes/no)?
no
/END/

STOP
```

Macro-processing

10.1/ Introduction

The macro-processing mechanism of QNAP2 is intended to facilitate the writing of frequently used sequences of commands or statements. It can also be used to define simple interfaces for dedicated models. The example described in this section illustrates these facilities.

10.2/ Description of the model

As in the first sections of this manual we consider a simple model of a transaction system. The following elements are represented in the model:

- the processing system: it is comprized of one central processing unit (cpu) and of a set of disk units;
- the workload: it consists of transactions organized into different classes; the transactions of a given class are issued from a set of terminals and are then processed by the processing system according to a behavior specific to their class.

The approach used in this example consists in defining a set of QNAP2 macro-statements, where each macro-statement contains the commands and statements associated with the description of the basic elements of the system studied and with their analysis. The following macro-statements will be defined:

- $init : initiation of a session,
- $cpu : definition or modification of the cpu caracteristics,
- $disk : definition or modification of a disk,
- $load : definition or modification of a load,
- $solve : analysis of the model,
- $edit : edition of the results,
- $end : end of a session.

10.3/ Definition of the macro-statements

10.3.1/ Macro-statement init

The function of this macro-statement is to perform the declaration of the object types and objects which will be used in the other macro-statements.

The declaration of the macro-statement is as follows:

```
$MACRO init
```

Then the data structures used for the elements of the models are defined. The concept of class of transaction will be represented by the predefined object type CLASS, with the following additionnal attributes:

- the number of terminals for this class of transactions,
- the mean thinking time at the terminals,
- the priority level at the cpu,
- the index of the class,
- the mean number of cpu instructions (in Mips) executed per transactions,
- the visit ratios to the disk units;

The declaration of the attributes is as follows:

```
/DECLARE/
&
CLASS INTEGER users, priority, index;
CLASS REAL thnktime, cputime;
CLASS REAL visit(25);
&
REF CLASS r_load(10);
```

The disk units will be defined by a new object type, disk, containing the following attributes:

- a queue,
- a real variable representing the disk mean service time.

```
OBJECT disk;
 QUEUE q_disk;
 REAL t_disk;
END;
&
REF disk r_disk(25); INTEGER nb_disk;
REF QUEUE r_q(25);
```

An array of 25 references to the object type disk is also declared. These references will be used to handle the object disk created later. The integer nb_disk will contain the number of disk units of the model. The array of queue references will be used to handle the queues associated with the disk units.

The central processing unit will be defined as an object of type QUEUE. The real variable rate represents the cpu instruction rate in Mips.

```
/DECLARE/
&
QUEUE cpu;
REAL rate;
```

The terminal station is defined as a queue with an infinite numbers of servers:

```
QUEUE terminal;
```

The stations associated with the cpu queue, the q_disk queues and the queue terminal are now defined.

The station associated with the queue q_disk of each object of type disk is defined with the command /STATION/. It is to be noted here that a "virtual" description has to be performed because no queue q_disk has yet been created. In order to do so a special notation with the "star" symbol is used in the NAME parameter indicating that the definition applies to a queue declared as attribute of an object type.

```
/STATION/ NAME = *q_disk;
          SERVICE = EXP(t_disk);
          TRANSIT = cpu;
&
/STATION/ NAME =cpu;
          SCHED = PRIOR, PREEMPT;
          PRIOR = priority;
          SERVICE = EXP(cputime);
          RATE = rate;
          TRANSIT = r_q(1 STEP 1 UNTIL nb_disk),
                    visit(1 STEP 1 UNTIL nb_disk),
                    terminal,1;
&
/STATION/ NAME = terminal;
          TYPE = INFINITE;
          SERVICE = EXP(thnktime);
          TRANSIT = cpu;
          INIT = users;
```

The definition of the macro-statement $init is completed by the declaration of working variables and specific options.

```
/DECLARE/ INTEGER i, j; REAL x; REF CLASS r_class;
&
/CONTROL/ OPTION = NRESULT; CLASS = ALL QUEUE; UNIT = GET(5);
```

It is then terminated as follows:

```
$END
```

10.3.2/ Macro-statement disk;

The function of the macro-statement $disk is to create a disk unit with is associated attribute.

```
$MACRO disk
&
/EXEC/ BEGIN
       PRINT(" ");
       PRINT("disk unit number :");
       i:= GET(INTEGER);
       IF i > nb_disk THEN nb_disk:= i;
       IF r_disk(i) = NIL THEN BEGIN
                               r_disk(i):= NEW(disk);
                               r_q(i):= r_disk(i).q_disk;
                               END;
       PRINT("disk service time:");
       r_disk(i).t_disk:= GET(REAL);
       END;
&
$END
```

In the macro-statement the existence of a disk unit with the given index is first checked. If the disk unit has not yet been created, it is then created dynamically by the procedure NEW. This causes also the creation of the associated station according to the description given before. Then the value of the mean service time is assigned (or reassigned if the disk unit was already created) to the attribute t_disk.

10.3.3/ Macro-statement cpu

The function of the macro-statement $cpu is to specify the characteristics of the processing unit of the system: number of cpu's and rate of instructions.

```
$MACRO cpu
&
/EXEC/ BEGIN
       PRINT(" ");
       PRINT("cpu rate (Mips) :");
       rate:= GET(REAL);
       END;
&
$END
```

10.3.4/ Macro-statement load

The function of the macro-statement $loads is to create an object of type load and to assign its attributes. The configuration of the system should be defined (cpu and disks) before the loads are defined.

```
$MACRO load
&
/EXEC/ BEGIN
       PRINT(" ");
       PRINT("load number          :");
       i:= GET(INTEGER);
       IF r_load(i) = NIL THEN r_load(i):= NEW(CLASS);
       r_load(i).index:= i;
       PRINT("number of users      :");
       r_load(i).users:= GET(INTEGER);
       PRINT("mean think time      :");
       r_load(i).thnktime:= GET(REAL);
       PRINT("priority level       :");
       r_load(i).priority:= GET(INTEGER);
       PRINT("cpu instructions (M):");
       r_load(i).cputime:= GET(REAL);
       FOR j := 1 STEP 1 UNTIL nb_disk DO
           BEGIN
           IF r_disk(j) <> NIL THEN
                  BEGIN
                  PRINT("visit ratio to disk ",j,":");
                  r_load(i).visit(j):= GET(REAL);
                  END;
           END;
       END;
&
$END
```

10.3.5/ Macro-statement solve

The function of the macro-statement $solve is to activate the analysis of the model specified with the previous macro-statement.

```
$MACRO solve
&
/EXEC/ SOLVE;
&
$END
```

10.3.6/ Macro-statement edit

The function of the macro-statement $edit is to edit the results of the analysis. The mean response time, throughput and cpu utilization will be edited by the macro-statement $edit for each classes and for the whole workload.

```
$MACRO edit
&
&
/EXEC/ BEGIN
       x:=0.0;
       PRINT(" ");
       PRINT("class cpu load   throughput    mean response time");
       FOR r_class:= ALL CLASS DO
          BEGIN
          x:= x+ r_class.users;
          IF MTHRUPUT(terminal,r_class) <> 0.0 THEN
                              PRINT(r_class.index, MBUSYPCT(cpu,r_class),
                                    MTHRUPUT(terminal,r_class),
                                    r_class.users/MTHRUPUT(terminal, r_class)-
                                    r_class.thnktime);
          END;
       PRINT("       all",MBUSYPCT(cpu), MTHRUPUT(terminal),
            x/MTHRUPUT(terminal));
       END;
&
$END
```

10.3.7/ Macro-statement end

The function of the macro-statement $end is to terminate a session:

```
$MACRO end
&
/END/
&
$END
```

10.4/ Library of macro-statements

The set of macro-statement are saved on a library file by using the SAVE procedure. The content of this file will be restored at the initiation of a session.

```
/EXEC/ SAVE("macro");
```

10.5/ Example of a session

A session is initiated by processing the following QNAP2 statements and commands.

```
/EXEC/ RESTORE("macro");
&
/TERMINAL/
```

The first command restores the library of macro-statements. The /TERMINAL/ command sets QNAP2 in the interactive mode so that the user can input the macro-statements from his terminal.

An example of a session is now given:

```
$init
$cpu

cpu rate (Mips) :
1
$disk

disk unit number :
1
disk service time:
0.08
$disk

disk unit number :
2
disk service time:
0.06
$load

load number         :
1
number of users     :
```

```
3
mean think time      :
10.0
priority level       :
1
cpu instructions (M):
0.5
visit ratio to disk        1.:
5
visit ratio to disk        2.:
5
$load

load number          :
2
number of users      :
5
mean think time      :
15.0
priority level       :
2
cpu instructions (M):
0.5
visit ratio to disk        1.:
3
visit ratio to disk        2.:
3
$solve
$edit

class cpu load   throughput    mean response time
 1.   .2183      0.3969E-01   65.59
 2.   .7646       .2184       7.889
all   .9829       .2581       30.99
```

The caracteristics of the cpu can be modified as follows:

```
 $cpu

cpu rate (Mips) :
1.4
 $solve
 $edit

class cpu load   throughput    mean response time
 1.   .3152      0.8024E-01   27.39
 2.   .6272       .2509       4.931
all   .9424       .3311       24.16
```

In the same fashion a new class of transactions can be added:

```
 $load

load number          :
3
number of users      :
8
mean think time      :
25
priority level       :
5
cpu instructions (M):
0.2
visit ratio to disk          1.:
2
visit ratio to disk          2.:
5
 $solve
 $edit
```

class	cpu load	throughput	mean response time
1.	.1395	0.3551E-01	74.48
2.	.5099	.2040	9.514
3.	.3373	.2951	2.110
all	.9867	.5346	29.93

It is to be noted that the QNAP2 commands and statements can nevertheless be used as usual. For example, detailed results can be required with the procedure OUTPUT.

```
 /EXEC/ OUTPUT;
*************************************************************************
*  NAME    *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*          *          *          *          *          *          *
* cpu      * .2361    * .9867    * 4.930    * 1.180    * 4.179    *
*(CLASS  1)* .3571    * .1395    * 2.617    * 6.699    * .3906    *
*(CLASS  2)* .3571    * .5099    * 1.844    * 1.292    * 1.428    *
*(CLASS  3)* .1429    * .3373    * .4689    * .1986    * 2.361    *
*          *          *          *          *          *          *
* terminal * 20.19    *0.0000E+00* 10.79    * 20.19    * .5346    *
*(CLASS  1)* 10.00    *0.0000E+00* .3551    * 10.00    *0.3551E-01*
*(CLASS  2)* 15.00    *0.0000E+00* 3.059    * 15.00    * .2040    *
*(CLASS  3)* 25.00    *0.0000E+00* 7.377    * 25.00    * .2951    *
*          *          *          *          *          *          *
*. *          *          *          *          *          *
* q_disk   *0.8000E-01* .1104    * .1229    *0.8909E-01* 1.380    *
*(CLASS  1)*0.8000E-01*0.1421E-01*0.1587E-01*0.8936E-01* .1776    *
*(CLASS  2)*0.8000E-01*0.4895E-01*0.5438E-01*0.8887E-01* .6119    *
*(CLASS  3)*0.8000E-01*0.4722E-01*0.5267E-01*0.8925E-01* .5902    *
*          *          *          *          *          *          *
*. *          *          *          *          *          *
* q_disk   *0.6000E-01* .1359    * .1553    *0.6858E-01* 2.265    *
*(CLASS  1)*0.6000E-01*0.1065E-01*0.1226E-01*0.6904E-01* .1776    *
*(CLASS  2)*0.6000E-01*0.3671E-01*0.4206E-01*0.6874E-01* .6119    *
*(CLASS  3)*0.6000E-01*0.8853E-01* .1010    *0.6846E-01* 1.475    *
*          *          *          *          *          *          *
*************************************************************************
```

The sesion is terminated by the macro-statement $end.

```
 $end

STOP
```

A model with internal concurrency

The example presented in this section is a queueing network model for programs with internal concurrency. This example shows how QNAP2 can be used to build new solution techniques from existing solvers and thus analyse models whose complexity prevents the use of standard techniques.

The model and its solution have been published in:

Heidelberger, P. and Trivedi, K. S. (1982). Analytic queueing models for programs with internal concurrency. IBM Research Report RC - 9194, Yorktown Heigths, New York.

11.1/ Presentation of the model

The system studied consists of a finite number, m, of processors and a finite number, n, of jobs. A job consists of a primary task and two or more secondary taks. Each primary task executes on a sequence of processors, this sequence of processors forming a Markov chain. There is a particular node, labeled 0, with 0 service time such that whenever a primary task enters this node it is split into two or more secondary tasks. A primary task is the parent of its secondary tasks and the latter are said to be siblings. The secondary tasks execute concurrently, except for queueing effects, independently of one another. Each secondary task execute a sequence of processors, the sequence of processors forming its own Markov chain. A secondary task is considered complete upon entering node 0. At node 0 the secondary task must wait until all of its siblings have completed execution at wich time the primary task becomes active again and the process is repeated. Synchronization between secondary tasks is achieved by requiring all siblings to complete execution before the job can continue processing.

A solution to this model consists of iteratively solving a sequence of product-form networks so that upon convergence, the solution to the product-form network closely approximate the solution of the system. Each of these product-form networks will have m+1 service stations (the m processors and a fictitious station) and d+1 classes of customers if the number of secondary tasks spawned by one primary task is d. The fictitious station is defined as follows:

Each secondary task is in one of the three macro-state: dormant, active, or waiting for the completion of its siblings. The primary task is in one of two states: active or waiting for all its secondary tasks to complete. In the active state a task consists of a sequence of visits to system resources and

hence is easily represented by a chain of the queueing network. The waiting state of a task will be represented by a fictitious delay type server, i.e. an infinite server station. Similarly, the dormant state will be represented by the delay server.

For sake of simplicity we make the following assumptions. The network of processor is of the central server type. The queueing discipline at the central processor is PS (processor-sharing); the queueing discipline at the other processors if FIFO. All service times have exponential distributions. The tasks may have different mean service times at the central processor but they have identical service times at the other processors. All tasks start on the central processor and a task cannot split or complete after executing the central processor.

11.2/ QNAP2 description of the model

The description and analysis of this model with QNAP2 will illustrate the object management facilities of QNAP2 and the use of QNAP2 for building and coding new solution algorithms over existing solvers.

11.2.1/ Description of the processor queues

In order to represent the processors and the fictitious station we add two additional attributes to the object type QUEUE: a real variable t representing the mean service time of tasks and an integer variable index. Then an array of 21 queues which will be used to manipulate the processors is created (the maximum number of processors in the model will be limited to 20).

```
/DECLARE/ QUEUE REAL t; QUEUE INTEGER index;

QUEUE proc(0:20);
```

11.2.2/ Description of the tasks

A task is specified as an object type defined with reference to the predefined type CLASS. The attributes of a task are:

- two vectors, transit1 and transit2, of transition probabilities,
- its mean service time tx at the central processor,
- a reference, parent, to its parent task,
- two real variables s0 and s1 representing its mean delay at the

fictitious station at two consecutive iterations of the iterative procedure,
- the number n of instances of this task,
- a string containing the name of the task.

The vector transit1 contains the transition probabilities of the task from the central processor (assumed processor number 1) to the processors 2, 3...,m. For a primary task the element transit2(i) contains the probability that the task splits after executing on the processor i (i > 1); for a secondary task the element transit2(i) contains the probability that the task completes after executing on the processor i (i > 1).

```
CLASS OBJECT task;
      INTEGER n;
      REAL transit1(2:20), transit2(2:20), tx, s0, s1;
      REF task parent;
      STRING name;
      END;

REF task r_task1, r_task2;
```

11.2.3/ Description of the stations processor

The service stations representing the processors and the fictitious station are now described. First, an INTEGER, m, representing the actual number of processors is declared and then the processors are described using the command /STATION/.

```
/DECLARE/ INTEGER m;

/STATION/ NAME = proc(2 STEP 1 UNTIL 20);
          SERVICE = EXP(t);
          TRANSIT = proc(0), transit2(QUEUE.index), proc(1);

/STATION/ NAME = proc(0);
          SERVICE = EXP(s0);
          TRANSIT = proc(1);
          TYPE = INFINITE;
          INIT = n;

/STATION/ NAME = proc(1);
          SERVICE = EXP(tx);
          TRANSIT = proc(2 STEP 1 UNTIL m),transit1(2 STEP 1 UNTIL m);
          SCHED = PS;
```

Note that the description of the processor stations is parametrized by m. The expressions appearing in the right hand side of the parameters of the /STATION/ command will be evaluated when a solver is activated (with the exception of the parameter NAME which is evaluated at compile time).

11.2.4/ Specification of a user defined procedure

The approximate method used requires the computation of the mean value of a random variable defined as the MAX of k independent exponential random variables. A specific procedure is defined to carry out this computation:

```
/DECLARE/ REAL x, z(50);

PROCEDURE emax ( k );
          INTEGER k, l; REAL b, c;
          BEGIN
```

for sake of brevity the complete description of the procedure emax is not given here. k is the number of independent random variables considered, z(k) is the mean of the k-th variable. The result is contained in the global variable x. l, b and c are local variables.

```
          END;
```

11.3/ Specification of an input dialogue

The following commands specify an input dialogue. First, working variables are declared and control parameters are set.

```
/DECLARE/ OBJECT answer; END;
          answer yes, no;

          LABEL def0, def1, def2, iter; INTEGER niter;

/CONTROL/ CLASS = ALL QUEUE; OPTION = NRESULT;
/CONTROL/ UNIT = GET(5);

/DECLARE/ INTEGER i, j; REAL epsilon =0.005;
```

Then an EXEC block specifies a dialogue for the input of the number of processors and the definion of the configuration of the system (characteristics of primary tasks, number and characteristics of secondary tasks for each type of primary tasks) and perform the analysis of the model:

```
/EXEC/ BEGIN

definition of the number of processors

 def0: PRINT("number of processors ?");
       m:= GET(INTEGER);
       IF m > 20 THEN BEGIN
                      PRINT("number of processors limited to 20. Try again");
                      GOTO def0;
                      END;

initialization of the processor queues index

       FOR i:=0 STEP 1 UNTIL m DO proc(i).index:= i;

definition of primary tasks

       PRINT(" ");
  def1:PRINT("definition of a primary task (yes or no)");
       IF GET(answer) = yes
       THEN BEGIN
            r_task1:= NEW(task);
            PRINT(" ");
            PRINT("name of the primary task ?");
            r_task1.name:= GET(STRING);
            PRINT("number of such primary tasks");
            r_task1.n:= GET(INTEGER);
            PRINT("mean service time at central processor");
            r_task1.tx:= GET(REAL);
            PRINT("transition probabilities from processor 1 to 2,..",
            m);
            FOR i:= 2 STEP 1 UNTIL m DO r_task1.transit1(i):= GET(REAL);
            PRINT("transition probabilities from processors 2,..",
            m," to node 0 (split)");
            FOR i:= 2 STEP 1 UNTIL m DO r_task1.transit2(i):= GET(REAL);

definition of secondary tasks

            PRINT(" ");
       def2:PRINT("definition of secondary task (yes or no)");
            IF GET(answer) = yes
            THEN BEGIN
                 r_task2:= NEW(task);
                 r_task2.parent:= r_task1;
                 PRINT(" ");
                 PRINT("name of the secondary task ?");
                 r_task2.name:= GET(STRING);
                 r_task2.n:= r_task1.n;
                 PRINT("mean service time at central processor");
                 r_task2.tx:= GET(REAL);
```

```
                PRINT("transition probabilities from processor 1  to 2,.,",
                m);
                FOR i:= 2 STEP 1 UNTIL m DO
                    r_task2.transit1(i):= GET(REAL);
                PRINT("transition probabilities from processors 2, ...",
                m," to node 0 (join)");
                FOR i:= 2 STEP 1 UNTIL m DO
                    r_task2.transit2(i):= GET(REAL);
                GOTO def2;
                END;
            GOTO def1;
            END;

        PRINT(" ");
        PRINT("mean service times at processors 2,...",m);
        FOR i:= 2 STEP 1 UNTIL m DO proc(i).t:= GET(REAL);

        FOR r_task1:= ALL task DO r_task1.s0:= r_task1.tx;

iterative solution

        niter:=1;

        NETWORK( proc( 0 STEP 1 UNTIL m) );

 iter: SOLVE;

        FOR r_task1:= ALL task DO
            BEGIN
            x:= 0.;
            FOR i:= 1 STEP 1 UNTIL m DO
                x:= x+ MCUSTNB(proc(i), r_task1);
            r_task1.s0:= x/MTHRUPUT(proc(0), r_task1);
            END;
        FOR r_task1:= ALL task WITH parent = NIL DO
            BEGIN
            i:= 0;
            FOR r_task2 := ALL task DO IF r_task2.parent = r_task1 THEN
                BEGIN
                i:= i+ 1;
                z(i):= r_task2.s0;
                END;
            emax(i);
            FOR r_task2 := ALL task DO IF r_task2.parent = r_task1 THEN
                r_task2.s0:=x- r_task2.s0+ r_task1.s0;
            r_task1.s0:= x;
            END;

convergence test
```

```
        i:= 0; x:= 0.;
        FOR r_task1:= ALL task DO
            BEGIN
            i:= i+1;
            x:= x+ (r_task1.s0- r_task1.s1)**2;
            r_task1.s1:= r_task1.s0;
            END;
        x:= (x**0.5)/i;
        IF x >= epsilon THEN BEGIN
                             niter:=niter+ 1;
                             GOTO iter;
                             END;

results

        PRINT(" ");
        PRINT("task   thruput      delay    proc 1 busy pct");
        PRINT(" ");
        FOR r_task1:= ALL task DO
            PRINT(r_task1.name, MTHRUPUT(proc(1), r_task1),
                  MSERVICE(proc(0), r_task1),
                  MBUSYPCT(proc(1), r_task1));

detailed results

        PRINT(" ");
        PRINT("detailed results requested (yes or no)");
        PRINT(" ");
        IF GET(answer) = yes THEN OUTPUT;

definition of a new configuration

        PRINT(" ");
        PRINT("definition of a new workload (yes or no)");
        IF GET(answer) = yes
        THEN BEGIN
             FOR r_task1:= ALL task DO r_task1.n:= 0;
             GOTO def0;
             END;
        END;
```

11.4/ Analysis of a model

The input parameters are now entered according to the dialogue defined. The configuration is made of 5 processors. The workload consists of 5 identical primary tasks, each primary task splitting into two secondary tasks, The mean service of all tasks at the central processor is 0.01; The mean service time of

all tasks at the other processors is 0.04; the probability that a primary task splits after executing on one of the processors 2, 3,..5 is 0.1; the probability that a secondary task completes is 0.1 for the first secondary task, 0.05 for the other one.

```
number of processors ?
   5
5

definition of a primary task (yes or no)
yes

name of the priimary task ?
"t1"
number of such primary tasks
5
mean service time at central processor
0.01
transition probabilities from processor 1 to 2,..        5.
0.25 0.25 0.25 0.25
transition probabilities from processors 2,..        5. to node 0 (split)
0.1 0.1 0.1 0.1

definition of secondary task (yes or no)
yes

name of the secondary task ?
"t11"
mean service time at central processor
0.01
transition probabilities from processor 1 to 2,..        5.
0.25 0.25 0.25 0.25
transition probabilities from processors 2, ...        5. to node 0 (join)
0.1 0.1 0.1 0.1
definition of secondary task (yes or no)
yes

name of the secondary task ?
"t12"
mean service time at central processor
0.01
transition probabilities from processor 1 to 2,..        5.
0.25 0.25 0.25 0.25
transition probabilities from processors 2, ...        5. to node 0 (join)
0.05 0.05 0.05 0.05
definition of secondary task (yes or no)
no
definition of a primary task (yes or no)
no
```

```
mean service times at processors 2,...        5.
0.04 0.04 0.04 0.04

task    thruput       delay     proc 1 busy pct

t1    14.45        2.403        .1445
t11    14.45         2.403         .1445
t12    28.94         1.423         .2894

detailed results requested (yes or no)

yes
*************************************************************************
*  NAME     *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *
*************************************************************************
*           *          *          *          *          *          *
* proc   1 * 2.076     *0.0000E+00* 9.006     * 2.076     * 4.338     *
*(task   1)* 2.403     *0.0000E+00* 3.473     * 2.403     * 1.445     *
*(task   2)* 2.403     *0.0000E+00* 3.473     * 2.403     * 1.445     *
*(task   3)* 1.423     *0.0000E+00* 2.059     * 1.423     * 1.447     *
*           *          *          *          *          *          *
* proc   2 *0.1000E-01* .5785     * 1.199     *0.2073E-01* 57.85     *
*(task   1)*0.1000E-01* .1445     * .3054     *0.2113E-01* 14.45     *
*(task   2)*0.1000E-01* .1445     * .3054     *0.2113E-01* 14.45     *
*(task   3)*0.1000E-01* .2894     * .5881     *0.2032E-01* 28.94     *
*           *          *          *          *          *          *
* proc   3 *0.4000E-01* .5785     * 1.199     *0.8290E-01* 14.46     *
*(task   1)*0.4000E-01* .1445     * .3054     *0.8452E-01* 3.613     *
*(task   2)*0.4000E-01* .1445     * .3054     *0.8452E-01* 3.613     *
*(task   3)*0.4000E-01* .2894     * .5881     *0.8129E-01* 7.235     *
*           *          *          *          *          *          *
* proc   4 *0.4000E-01* .5785     * 1.199     *0.8290E-01* 14.46     *
*(task   1)*0.4000E-01* .1445     * .3054     *0.8452E-01* 3.613     *
*(task   2)*0.4000E-01* .1445     * .3054     *0.8452E-01* 3.613     *
*(task   3)*0.4000E-01* .2894     * .5881     *0.8129E-01* 7.235     *
*           *          *          *          *          *          *
* proc   5 *0.4000E-01* .5785     * 1.199     *0.8290E-01* 14.46     *
*(task   1)*0.4000E-01* .1445     * .3054     *0.8452E-01* 3.613     *
*(task   2)*0.4000E-01* .1445     * .3054     *0.8452E-01* 3.613     *
*(task   3)*0.4000E-01* .2894     * .5881     *0.8129E-01* 7.235     *
*           *          *          *          *          *          *
* proc   6 *0.4000E-01* .5785     * 1.199     *0.8290E-01* 14.46     *
*(task   1)*0.4000E-01* .1445     * .3054     *0.8452E-01* 3.613     *
*(task   2)*0.4000E-01* .1445     * .3054     *0.8452E-01* 3.613     *
*(task   3)*0.4000E-01* .2894     * .5881     *0.8129E-01* 7.235     *
*           *          *          *          *          *          *
*************************************************************************
```

```
definition of a new workload (yes or no)
no
 /END/

STOP
```

A communication network model

12.1/ Presentation

The example presented in this section is a general communication network model. It illustrates how the language of QNAP2 can be used to build customized environments and user's interface. In this sense QNAP2 can be viewed as a sytem for developping dedicated performance prediction tools to be used by users with no experience in modelling techniques.

12.2/ Description of the communication network model

We consider a general model of a packet switching communication network. The model is caracterized by the following elements:

- the network topology:
  - + switching nodes,
  - + lines,
  - + processing nodes,
  - + source nodes,
- the flows of packets:
  - + origin and destination nodes,
  - + routing of packets,
  - + packet and acknowlegment size,

12.3/ Definition of a specific environment

The first step in building a QNAP2 model is to define the types of the objects which are needed to represent the elements of the system considered. In many situations, it is helpful to define new object types having as attributes the existing predefined object types QUEUE, CLASS, CUSTOMERS and FLAG and scalar or dimensioned variables. In this way data structures representing complex entities can be created and manipulated in a flexible and efficient fashion.

For instance, in the example studied, we will define new object types in order to represent the basic elements of a communication network.

12.3.1/ Switching nodes

In order to represent switching nodes, a new object type, node, is declared. A node is made of a queue q_node, a string of characters containing the name of the node, a real scalar switch representing the mean switching time of messages and an index:

```
/DECLARE/

OBJECT node (name_nd, switch, index_nd);

        QUEUE q_node;
        STRING name_nd;
        REAL switch;
        INTEGER index_nd;
        END;
```

Note that the object type node is a parametrized object type. The parameters will be used to facilitate attribute initialization at object creation time.

12.3.2/ Specification of the number of nodes

We define here a simple dialogue with the user in order to enter the number of nodes of the model considered:

```
/DECLARE/ INTEGER n_node, getunit = 5;

/CONTROL/ UNIT = GET(getunit);

/EXEC/ BEGIN
       PRINT("number of nodes ?");
       n_node:= GET(INTEGER);
       END;
```

```
number of nodes ?

5
```

The next declaration creates an array of node references which will be used to manipulate the nodes.

```
/DECLARE/ REF node r_node(n_node);
```

12.3.3/ Lines

We define a new object type, line, to represent the physical lines of the network. The object type line is comprised of a queue, the name of the line, a real rate giving the transmission rate of the line and of the indices of the nodes linked by the line.

```
OBJECT line (name_ln, node1, node2, rate);

      QUEUE q_line; STRING name_ln;
      REAL rate; INTEGER node1, node2;
      END;
```

The next declaration defines an array of queue references. The entry (i,j) of this array will contain a reference to the queue of the line linking node i and node j if this line exists; a null reference if there is no line between node i and node j.

```
REF QUEUE routage(0:n_node, 0:n_node);
```

12.3.4/ Processing nodes

The processing nodes are defined as objects with the new object type center. A center is comprised of a queue, a name and a real scalar representing the mean time to process a packet at the destination center before sending an acknowledgement.

```
OBJECT center (name_ct, process);
       QUEUE q_center;
       STRING name_ct;
       REAL process;
       END;
```

12.3.5/ Flows

The data structure associated with a flow is defined as follow:

A flow is comprised of two path: the path of packets and the path of corresponding acknowledgements. A path is specified as an object type referencing the predefined type CLASS and has the following additional attributes: the indices of the first and the last node of the path, a routing vector desti where desti(j) contains the index of the node following node j, size of the messaages (packets or acknowledgements), a reference to the output queue of the path and a reference to the associated path.

```
/DECLARE/

CLASS OBJECT path;
      INTEGER desti(n_node), first, last;
      REAL size;
      REF QUEUE out;
      REF path other;
      END;
```

The object type flow is made of its two path and a name:

```
OBJECT flow;

INTEGER index_fl;
STRING name_fl;
path packet, ack;
END;
```

12.3.6/ Source nodes

Packets are sent by sources. A source is defined as a new object type source with the following attributes: a queue, a reference to the flow associated with this source and a scalar load representing the mean arrival rate of packets in the network.

```
/DECLARE/

OBJECT source (r_flow, load);
      QUEUE q_source;
      REF flow r_flow;
      REAL load;
      END;
```

12.4/ Specification of the the service stations

The second step in the specification of the general model considered is the specification of the service stations associated with the queue q_node, q_center, q_source, q_line.

A powerfull facility associated with an object type which is comprized of one or more objects of type QUEUE as attributes is the possibility to define a template station in association with each of these queues. The creation of an object of type considered will then result in the creation of the associated stations.

We now define the template stations associated with the queue attributes of the object types node, center, line and source. This is done by the standard /STATION/ command with the parameter NAME equal to the queue identifier prefixed by a "*":

```
/STATION/ NAME = *q_node;
          SERVICE = EXP(switch);
          TRANSIT = IF index_nd = last THEN out
                                ELSE routage (index_nd, desti(index_nd));

/STATION/ NAME = *q_center;
          SERVICE = EXP(process);
          TRANSIT = r_node(other.first).q_node, other;

/STATION/ NAME = *q_line;
          SERVICE = EXP(size/rate);.
          TRANSIT = r_node(node2).q_node;

/STATION/ NAME =*q_source;
          TYPE = SOURCE;
          SERVICE = EXP(1./load);
          TRANSIT = r_node(r_flow.packet.first).q_node, r_flow.packet;
```

The form of the TRANSIT parameters illustrates the usage of complex expressions in the parameters of the command /STATION/. In these expressions an extensive usage is made of the attributes of the current queue and the current class. For example, in the TRANSIT parameter of the command /STATION/ describing the station associated with the queue q_node, index_nd implicitly refers to the queue being currently described and desti refers to the current class for which the TRANSIT parameter is evaluated (note that the parameters of the command will be evaluated for all the declared classes).

12.5/ Definition of a user interface

The elements for the construction of communication network models have been defined in the two previous sections. These elements can be in a third step created and organized on demand depending on the user's requirements. This can be done according to a predefined dialogue which can be specified using the facilities of the algorithmic language of QNAP2.

The next commands and statements define an interface for the data input. The different objects involved in the model are created on demand by using the procedure NEW.

```
/CONTROL/ OPTION = NRESULT; CLASS = ALL QUEUE;

/DECLARE/ INTEGER i, j, k, k0, k1, nb_line, nb_cent;

          REF line r_ln; REF node r_nd; REF flow r_fl; REF center r_ct;
          REF source r_source;

          LABEL b_ln, e_ln, b_fl, e_fl, b_lsa, e_lsa, b_lsp, e_lsp,
                b_ct, e_ct, ok_ct, ko_ct, ko_lsa, ko_lsp;

          STRING st;

          OBJECT response;END; response yes, no;

/EXEC/ BEGIN
       PRINT(" ");
       PRINT("switching nodes characteristics ?");
       PRINT(" ");

       FOR i:= 1 STEP 1 UNTIL n_node DO
           BEGIN
           PRINT("node ", i);
           PRINT(" name   switching time?");
           r_nd:= NEW(node, GET(STRING), GET(REAL), i);
           r_node(i):= r_nd;
           END;
```

```
        i:=0;
  b_ct:PRINT(" ");
        PRINT("processing center definition ? (yes or no)");
        PRINT(" ");
        IF GET(response) = no THEN GOTO e_ct;
        i:= i+1;
        PRINT("center ",i);
        PRINT("name    packet processing time ");
        r_ct:= NEW(center, GET(STRING), GET(REAL));
        GOTO b_ct;
  e_ct:nb_cent:= i;


        i:=0;
  b_ln:PRINT(" ");
        PRINT("line definition ? (yes or no)");
        PRINT(" ");
        IF GET(response) = no THEN GOTO e_ln;
        i:= i+1;
        PRINT("line ",i);
        PRINT(" name   input node   output node     rate (Bytes/sec.) ");
        st:= GET(STRING);
        j:= GET(INTEGER);
        k:= GET(INTEGER);
        r_ln:= NEW(line, st, j, k, GET(REAL));
        routage(j, k):= r_ln.q_line;
        GOTO b_ln;
e_ln:nb_line:= i;

        i:= 0;
  b_fl:PRINT(" ");
        PRINT("flow definition ? (yes or no)");
        PRINT(" ");
        IF GET(response) = no THEN GOTO e_fl;
        i:= i+1;
        PRINT("flow ",i);
        PRINT("name  arrival rate ");
        r_fl:= NEW(flow);
        r_fl.name_fl:= GET(STRING);
        r_source:= NEW(source, r_fl, GET(REAL));
  ko_ct:PRINT("center name");
        st:= GET(STRING);
        FOR r_ct:= ALL center DO IF r_ct.name_ct = st THEN GOTO ok_ct;
        PRINT("center ",st," does not exist.  Try again");
        GOTO ko_ct;
  ok_ct:r_fl.index_fl:=i;
        r_source.r_flow:= r_fl;
        r_fl.packet.out:= r_ct.q_center;
```

```
        r_fl.ack.out:= OUT;
        r_fl.packet.other:= r_fl.ack;
        r_fl.ack.other:= r_fl.packet;

ko_lsp:PRINT(" ");

PRINT("indices of nodes visited by forward flow of packets (o ends input)");
        j:= 0; kO:= 0;
b_lsp:j:= j+1;
        k:= GET(INTEGER);
        IF k = 0 THEN GOTO e_lsp;
        IF j =1 THEN k1:= k;
        IF j > 1 THEN BEGIN
                        IF routage(kO, k) = NIL THEN
                                BEGIN
                                PRINT("no line between nodes",kO," and ", k);
                                PRINT("Try again");
                                GOTO ko_lsp;
                                END;
                        r_fl.packet.desti(kO):= k;
                        END;
        kO:= k;
        GOTO b_lsp;

  e_lsp:r_fl.packet.first:= k1;
        r_fl.packet.last:= kO;
        PRINT(" ");
        PRINT("packet size (Bytes) ?");
        r_fl.packet.size:= GET(REAL);

ko_lsa:PRINT(" ");
PRINT("indices of nodes visited by backward flow of acks (o ends input)");
        j:= 0; kO:= 0;
b_lsa:j:= j+1;
        k:= GET(INTEGER);
        IF k = 0 THEN GOTO e_lsa;
        IF j =1 THEN k1:= k;
        IF j > 1 THEN BEGIN
                        IF routage(kO, k) = NIL THEN
                                BEGIN
                                PRINT("no line between nodes",kO," and ", k);
                                PRINT("Try again");
                                GOTO ko_lsa;
                                END;
                        r_fl.ack.desti(kO):= k;
                        END;
        kO:= k;
        GOTO b_lsa;

e_lsa:r_fl.ack.first:= k1;
```

```
        r_fl.ack.last:= kO;
        PRINT(" ");
        PRINT("ack size (Bytes) ?");
        r_fl.ack.size:= GET(REAL);

        PRINT(" ");
        GOTO b_fl;

e_fl:PRINT(" ");

FOR i:= 0 STEP 1 UNTIL n_node DO
FOR j:= 0 STEP 1 UNTIL n_node DO
IF routage(i,j) = NIL THEN routage (i,j):= OUT;

        SOLVE;

 edition des resultats

        PRINT(" ");
        PRINT("results on nodes");
        PRINT(" ");
        PRINT("node  busy        mean number total");
        PRINT("name  percentage  of packets  thruput");
        PRINT(" ");
        FOR r_nd:= ALL node DO
           PRINT(r_nd.name_nd, MBUSYPCT(r_nd.q_node), MCUSTNB(r_nd.q_node),
             MTHRUPUT(r_nd.q_node));
        PRINT(" ");
        PRINT("results on lines");
        PRINT(" ");
        PRINT("line  busy        mean number total");
        PRINT("name  percentage  of packets  thruput");
        PRINT(" ");
        FOR r_ln:= ALL line DO
           PRINT(r_ln.name_ln, MBUSYPCT(r_ln.q_line), MCUSTNB(r_ln.q_line),
                MTHRUPUT(r_ln.q_line));
        PRINT(" ");


        PRINT("detailed results ?(yes or no)");
        PRINT(".qnb");
        IF GET(response) = yes THEN OUTPUT;
        PRINT(".qne");

        PRINT(" ");
        PRINT("new arrival rates  ?(yes or no)");
        IF GET(response) = yes THEN
            BEGIN
            FOR r_source:= ALL source DO
                BEGIN
```

```
            PRINT(r_source.r_flow.name_fl," arrival rate ?");
            getunit := 4;
            r_source.load:= GET(REAL);
            END;
        GOTO e_fl;
        END;
END;
```

12.6/ Analysis of a model

We now enter the description of a communication network model. The model considered here is represented in the Figure 7. It consists of 5 nodes, 10 lines, 3 processing centers and 3 flows.

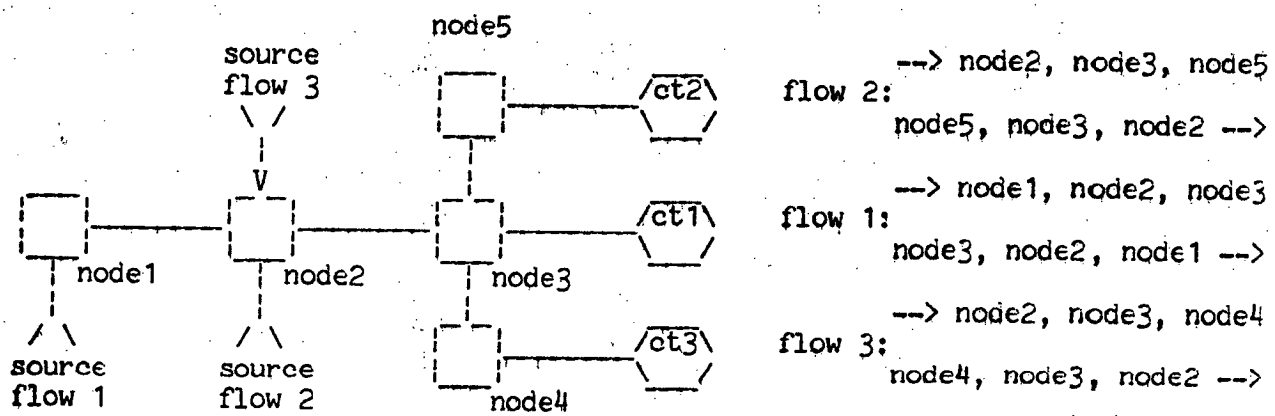

Figure 7

The input dialogue corresponding to this model is as follows:

```
switching nodes characteristics ?

node          1.
 name    switching time?
"node1" 0.01
node          2.
 name    switching time?
"node2" 0.01
node          3.
 name    switching time?
"node3" 0.01
node          4.
 name    switching time?
"node4" 0.01
node          5.
 name    switching time?
"node5" 0.01

processing center definition ? (yes or no)

yes
center          1.
name     packet processing time
"ct1" 0.03

processing center definition ? (yes or no)

yes
center          2.
name     packet processing time
"ct2" 0.025

processing center definition ? (yes or no)

yes
center          3.
name     packet processing time
"ct3" 0.03

processing center definition ? (yes or no)

no
```

```
line definition ? (yes or no)

yes
line          1.
 name    input node    output node      rate (Bytes/sec.)
"112" 1 2  120

line definition ? (yes or no)

yes
line          2.
 name    input node    output node      rate (Bytes/sec.)
"121" 2 1 120

line definition ? (yes or no)

yes
line          3.
 name    input node    output node      rate (Bytes/sec.)
"123" 2 3 960

line definition ? (yes or no)

yes
line          4.
 name    input node    output node      rate (Bytes/sec.)
"132" 3 2 960

line definition ? (yes or no)

yes
line          5.
 name    input node    output node      rate (Bytes/sec.)
"134" 3 4 480

line definition ? (yes or no)

yes
line          6.
 name    input node    output node      rate (Bytes/sec.)
"143" 4 3 480

line definition ? (yes or no)

yes
line          7.
 name    input node    output node      rate (Bytes/sec.)
"135" 3 5 240

line definition ? (yes or no)
```

```
yes
line          8.
 name    input node    output node      rate (Bytes/sec.)
"153" 5 3 240

line definition ? (yes or no)

no

flow definition ? (yes or no)

yes
flow          1.
name  arrival rate
"f1" 3.
center name
"ct1"

indices of nodes visited by forward flow of packets (o ends input)
1 2 3 0

packet size (Bytes) ?
20

indices of nodes visited by backward flow of acks (o ends input)
3 2 1 0

ack size (Bytes) ?
4
```

```
flow definition ? (yes or no)

yes
flow         2.
name  arrival rate
"f2" 5.
center name
"ct2"

indices of nodes visited by forward flow of packets (o ends input)
2 3 5 0

packet size (Bytes) ?
10

indices of nodes visited by backward flow of acks (o ends input)
5 3 2 0

ack size (Bytes) ?
4


flow definition ? (yes or no)

yes
flow         3.
name  arrival rate
"f3" 6.
center name
"ct3"

indices of nodes visited by forward flow of packets (o ends input)
2 3 4 0

packet size (Bytes) ?
30

indices of nodes visited by backward flow of acks (o ends input)
4 3 2 0

ack size (Bytes) ?
4


flow definition ? (yes or no)

no
```

```
results on nodes

node  busy          mean number  total
name  percentage    of packets   thruput

node1  0.6000E-01   0.6382E-01    6.000
node2    .2800        .3761      28.00
node3    .2800        .3886      28.00
node4    .1200        .1364      12.00
node5    .1000        .1111      10.00

results on lines

line  busy          mean number  total
name  percentage    of packets   thruput

112    .5000        .9993        3.000
121    .1000        .1110        3.000
123    .3021        .4520        14.00
132  0.5833E-01   0.6194E-01     14.00
134    .3750        .5999        6.000
143  0.5000E-01   0.5263E-01     6.000
135    .2083        .2631        5.000
153  0.8333E-01   0.9091E-01     5.000

detailed results ?(yes or no)
```

yes

NAME	SERVICE	BUSY PCT	CUST NB	RESPONSE	THRUPUT
node 1.					
q_node	*0.1000E-01	*0.6000E-01	*0.6382E-01	*0.1064E-01	6.000
(packet	)*0.1000E-01	*0.3000E-01	*0.3191E-01	*0.1064E-01	3.000
(ack	)*0.1000E-01	*0.3000E-01	*0.3191E-01	*0.1064E-01	3.000
node 2.					
q_node	*0.1000E-01	.2800	.3761	*0.1343E-01	28.00
(packet	)*0.1000E-01	*0.3000E-01	*0.4030E-01	*0.1343E-01	3.000
(ack	)*0.1000E-01	*0.3000E-01	*0.4030E-01	*0.1343E-01	3.000
(packet	)*0.1000E-01	*0.5000E-01	*0.6717E-01	*0.1343E-01	5.000
(ack	)*0.1000E-01	*0.5000E-01	*0.6717E-01	*0.1343E-01	5.000
(packet	)*0.1000E-01	*0.6000E-01	*0.8060E-01	*0.1343E-01	6.000
(ack	)*0.1000E-01	*0.6000E-01	*0.8060E-01	*0.1343E-01	6.000
node 3.					
q_node	*0.1000E-01	.2800	.3886	*0.1388E-01	28.00
(packet	)*0.1000E-01	*0.3000E-01	*0.4163E-01	*0.1388E-01	3.000
(ack	)*0.1000E-01	*0.3000E-01	*0.4163E-01	*0.1388E-01	3.000
(packet	)*0.1000E-01	*0.5000E-01	*0.6938E-01	*0.1388E-01	5.000
(ack	)*0.1000E-01	*0.5000E-01	*0.6938E-01	*0.1388E-01	5.000
(packet	)*0.1000E-01	*0.6000E-01	*0.8326E-01	*0.1388E-01	6.000
(ack	)*0.1000E-01	*0.6000E-01	*0.8326E-01	*0.1388E-01	6.000
node 4.					
q_node	*0.1000E-01	.1200	.1364	*0.1136E-01	12.00
(packet	)*0.1000E-01	*0.6000E-01	*0.6818E-01	*0.1136E-01	6.000
(ack	)*0.1000E-01	*0.6000E-01	*0.6818E-01	*0.1136E-01	6.000
node 5.					
q_node	*0.1000E-01	.1000	.1111	*0.1111E-01	10.00
(packet	)*0.1000E-01	*0.5000E-01	*0.5555E-01	*0.1111E-01	5.000
(ack	)*0.1000E-01	*0.5000E-01	*0.5555E-01	*0.1111E-01	5.000
cente 1.					
q_center	*0.3000E-01	*0.9000E-01	*0.9890E-01	*0.3297E-01	3.000
(packet	)*0.3000E-01	*0.9000E-01	*0.9890E-01	*0.3297E-01	3.000
cente 2.					
q_center	*0.2500E-01	.1250	.1429	*0.2857E-01	5.000
(packet	)*0.2500E-01	.1250	.1429	*0.2857E-01	5.000
cente 3.					
q_center	*0.3000E-01	.1800	.2195	*0.3658E-01	6.000
(packet	)*0.3000E-01	.1800	.2195	*0.3658E-01	6.000

```
*line   1. *           *           *           *           *           *
* q_line   * .1667    * .5000     * .9993     * .3331     * 3.000     *
*(packet  )* .1667    * .5000     * .9993     * .3331     * 3.000     *
*          *           *           *           *           *           *
*line   2. *           *           *           *           *           *
* q_line   *0.3333E-01* .1000     * .1110     *0.3701E-01* 3.000     *
*(ack     )*0.3333E-01* .1000     * .1110     *0.3701E-01* 3.000     *
*          *           *           *           *           *           *
*line   3. *           *           *           *           *           *
* q_line   *0.2158E-01* .3021     * .4520     *0.3228E-01* 14.00     *
*(packet  )*0.2083E-01*0.6250E-01*0.9462E-01*0.3154E-01* 3.000     *
*(packet  )*0.1042E-01*0.5208E-01* .1056     *0.2112E-01* 5.000     *
*(packet  )*0.3125E-01* .1875     * .2517     *0.4196E-01* 6.000     *
*          *           *           *           *           *           *
*line   4. *           *           *           *           *           *
* q_line   *0.4167E-02*0.5833E-01*0.6194E-01*0.4424E-02* 14.00     *
*(ack     )*0.4167E-02*0.1250E-01*0.1327E-01*0.4424E-02* 3.000     *
*(ack     )*0.4167E-02*0.2083E-01*0.2212E-01*0.4424E-02* 5.000     *
*(ack     )*0.4167E-02*0.2500E-01*0.2655E-01*0.4424E-02* 6.000     *
*          *           *           *           *           *           *
*line   5. *           *           *           *           *           *
* q_line   *0.6250E-01* .3750     * .5999     *0.9998E-01* 6.000     *
*(packet  )*0.6250E-01* .3750     * .5999     *0.9998E-01* 6.000     *
*          *           *           *           *           *           *
*line   6. *           *           *           *           *           *
* q_line   *0.8333E-02*0.5000E-01*0.5263E-01*0.8772E-02* 6.000     *
*(ack     )*0.8333E-02*0.5000E-01*0.5263E-01*0.8772E-02* 6.000     *
*          *           *           *           *           *           *
*line   7. *           *           *           *           *           *
* q_line   *0.4167E-01* .2083     * .2631     *0.5263E-01* 5.000     *
*(packet  )*0.4167E-01* .2083     * .2631     *0.5263E-01* 5.000     *
*          *           *           *           *           *           *
*line   8. *           *           *           *           *           *
* q_line   *0.1667E-01*0.8333E-01*0.9091E-01*0.1818E-01* 5.000     *
*(ack     )*0.1667E-01*0.8333E-01*0.9091E-01*0.1818E-01* 5.000     *
*          *           *           *           *           *           *
*sourc  1. *           *           *           *           *           *
* q_source * .3333    * 1.000     * 1.000     * .3333     * 3.000     *
*          *           *           *           *           *           *
*sourc  2. *           *           *           *           *           *
* q_source * .2000    * 1.000     * 1.000     * .2000     * 5.000     *
*          *           *           *           *           *           *
*sourc  3. *           *           *           *           *           *
* q_source * .1667    * 1.000     * 1.000     * .1667     * 6.000     *
*          *           *           *           *           *           *
*************************************************************************

new arrival rates ?(yes or no) no

STOP
```

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique